[B/D] Reference Manual - Version 8.44

David Wooff
University of Durham

[B/D] Home Page: http://fourier.dur.ac.uk:8000/stats/bd/

Contents

Preface

This is the manual for the [B/D] programming language. [B/D] is the computer implementation of the Bayes linear methodology developed by Michael Goldstein and myself, initially at the University of Hull, and more recently at the University of Durham. Part of the research and development underlying the program has been funded by the Science and Engineering Research Council, in its various guises over the years, and we are extremely grateful to them for their support.

The manual assumes mostly that you know what Bayes linear methods are, and also that you are conversant with the basic notation used particularly in the two tutorial documents [31] and [48]. A third tutorial document, [49], illustrates using [B/D] to set up and analyse problems involving exchangeable situations, together with details on the creation and interpretation of Bayes linear influence diagrams. Hence, this manual does not tell you what analysis you should perform, but it should hopefully show you how you might go about performing the analysis that you have decided upon.

For newcomers to Bayes linear methods, a comprehensive bibliography is included at the rear of the manual. For the development of the theory, begin with [14] and read on! General overviews of some of the features in the package are given in [47] and [35].

There are different versions of the [B/D] package. For IBM PCs and compatibles there are WINDOWS versions, for machines with and without a co-processor. Further versions are available for running under UNIX and Linux. Our initial test release of [B/D] to the academic community consists of versions which run as Microsoft Windows 3.1 or Windows 95 applications.

The program itself can be regarded as complete and reasonably well (but not extensively) tested. Inevitably bugs will arise as the program is used to handle applications other than our own test macros and applications. Please let us know of any problems or oddities encountered. Further, we may develop or amend features as time permits.

David Wooff
July 1997

Chapter 1

Installation and interface

1.1 Overview

[B/D] is currently available in three versions: for IBM PCs and compatibles running under WINDOWS or Linux, and for the SUN workstations, running under UNIX, networked in the Department of Mathematical Sciences at Durham University. The last allows the analysis of larger problems, but this is essentially the only difference between the UNIX and other versions. The graphics handling for the three versions differs slightly.

We consider separately installing the PC version, and using the UNIX version. The Linux version is under construction. In the final part of this chapter we survey briefly the [B/D] interface: how command lines are constructed, and so forth.

1.1.1 Installing the WINDOWS version

The program needs access to the following files.

bd.cmd A text file defining [B/D]'s commands and other keywords. This file is essential - [B/D] will not work without it.

bd.msg A text file containing runtime error and warning messages. [B/D] will work if this file is missing, but no error messages will be available.

bd.hlp A large text file containing online help for the program. [B/D] will work if this file is missing, but no help will be available.

bd.exe The main [B/D] executable program;

Suppose that all the files listed above for a particular implementation are in your current directory, and that you plan to work solely within this directory. Then you can run the program by typing the command "bd" at your system prompt. Otherwise, you can configure the program so that it finds its files elsewhere, as follows.

We refer to the directory from which [B/D] is run as the current directory. All temporary files are placed on, and all output directed to, the current directory. The required files must be present either in the current directory, or in the path given by the environment variable BDINPUTS. The current directory is always searched first. (The search path can be added to interactively if necessary.) The environment variable can be set by adding, for example, the following line to your autoexec.bat file:

```
set BDINPUTS=C:\BD\BDNEEDS\
```

where `c:\BD\BDNEEDS` is the name chosen for the subdirectory containing the required files.

The BDMACROS environment variable is set in a similar way to the BDINPUTS variable. It can be set to contain a string of paths which are to be searched for input files. For example, you could add the following line to your autoexec.bat file:

```
set BDMACROS=c:\BD\BDINPUTS;D:\
```

For its input files, the effect here would be for [B/D] to search (by default) the current directory first; and then the subdirectory `c:\bd\bdinputs`, and so forth.

Starting the program

To start the program, double-click on the file "bd.exe" in File Manager (Windows 3.1) or Windows Explorer (Windows 95). A command window will appear, in which all input and text output takes place. In this command window, the following [B/D] prompt should appear:
`BD>`
indicating that the [B/D] program is running and awaiting your commands. The prompts are fully described in §20.13.

1.1.2 Installing the UNIX version

The UNIX version is already installed on the SUN workstations networked in the Department of Mathematical Sciences, University of Durham. You can run the program by typing the command "bd" at the unix prompt.

We refer to the directory from which [B/D] is run as the current directory. All temporary files are placed on, and all output directed to, the current directory. The various required files must be present either in the current directory, or in the path given by the environment variable BDINPUTS. The current directory is always searched first. (The search path can be added to interactively if necessary.)

For the version of [B/D] which is set up to run over the Durham University Mathematical Sciences Department network, this variable should be set by adding the following line to your .cshrc file:

```
setenv BDINPUTS /usr/local/src/modsrc/BD/
```

The BDMACROS environment variable is set in a similar way to the BDINPUTS variable. It can be set to contain a string of paths which are to be searched for input files. For example, you might add the following line to your .cshrc file:

```
setenv BDMACROS $HOME:$HOME/bd/bdinputs/
```

For its input files, the effect here would be for [B/D] to search (by default) the current directory first; and then the subdirectory `bd/bdinputs/`, and so forth.

Starting the program

To start the program, issue the "bd" command at your system prompt:

```
bd
```

The following [B/D] prompt should appear:
`BD>`
indicating that the [B/D] program is running and awaiting your commands. The prompts are fully described in §20.13.

1.2 Stopping the program

To stop the program, issue the STOP: command at the [B/D] prompt as follows. The '↩' symbol at the end of a lion means that you should press the <RETURN> or <ENTER> key.

`BD>`**stop:** ↩

1.3 Associating files and search paths with the program at start-up

1.3.1 Overview

When the program is run, input and output files and additions to the search path may be included on the command line, and it is possible to set up a file to act as a log-in file, so that [B/D] obeys the commands on the log-in file first. Input and output files can be attached, and additions to the search path made, interactively as well. We discuss each possibility in turn below; the various possibilities can be mixed on the eventual command line.

1.3.2 Attaching input files

To attach input files, typically containing [B/D] program macros, data, and so forth, via the command line we could start the program as follows:

```
bd [i1=file1] [i2=file2] ...
```

There are 6 input channels, i1 ... i6, which can be associated with an external file in this way. Input files may be located in any directory that will be searched according to the search path (which may also be amended at start-up: see below). The current directory is always searched first, and then the path given in the environment variable BDMACROS (if any). The BDMACROS environment variable is set as described above. It can be set to contain a string of paths which are to be searched for input files. The CHANNEL: command can also be used to associate input files with [B/D] channels.

If input channel i1 is assigned on the command line, it acts as a log-in channel: [B/D] obeys all commands therein in the usual way, including macro switches, and the end of file marker causes control to be returned to the screen channel. Otherwise channels can be assigned using the CHANNEL: command.

1.3.3 Attaching output files

To attach output files via the command line we could start the program as follows:

```
bd [o1=file1] [o2=file2] ...
```

There are 4 output channels, o1 ... o4, which can be associated with an external file in this way. The CHANNEL: command can also be used to associate output files with [B/D] channels. All output (standard output, a log of the current session if required, and a history of the screen input if required) is output to files in the current directory. The program will overwrite pre-existing output files, and otherwise create them.

1.3.4 Updating search paths

Temporary search paths can be given on the [B/D] command line or via the PATH: command. These are given as in the following examples. Firstly, for the UNIX version,

```
bd p=bd/bdinputs/special/:bd/oldinputs
```

and secondly for the WINDOWS version,

```
bd p=bd\bdinputs\special\;bd\oldinputs
```

These extra paths are temporarily (for the life of the session) added to the BDMACROS variable, and are searched immediately after the current directory. If files are also given on the command line, the path is processed first. The PATH: command can be used in a similar fashion.

1.4 The [B/D] input interface

1.4.1 Command lines

[B/D] acts upon separate lines of input, where a line of input is ended typically by pressing the ENTER key once, and consists of no more than 253 alphanumeric characters. (It is possible both to extend a [B/D] *command line* over more than one physical line, as we consider in §1.5, and to force more than one [B/D] *command line* onto one physical line, as we consider in §1.4.3. However, the maximum of 253 characters remains in force whatever the circumstances.) The legal alphanumeric characters are shown in Table 1.1. Otherwise, any illegal characters or invisible control characters supplied to [B/D] are treated as though they are spaces.

Each line of input is treated either as a *command line*, where [B/D] is intended to interpret a command and act accordingly, or not. The first non-space character of a line determines which is the case. Generally, all lines of input are assumed to be *command lines* unless the first non-space character is one of the following:

- a digit or a minus sign, implying that the line consists of numbers;

Table 1.1: Characters allowed as [B/D] input

<table>
<tr><td>The lower case characters</td><td>a, . . . , z</td></tr>
<tr><td>The upper case characters</td><td>A, . . . , Z</td></tr>
<tr><td>The digits</td><td>0, . . . , 9</td></tr>
<tr><td>The parenthetic characters</td><td>( ) { } [ ]</td></tr>
<tr><td>The arithmetic characters</td><td>+ - / * ^ > < =</td></tr>
<tr><td>The characters</td><td>! " # & % $ & _ ~ : ; @ | \ , . ?</td></tr>
<tr><td>The space</td><td></td></tr>
</table>

- the symbol '@', implying that the line contains a *label* or commentary text that [B/D] should not interpret. Any text between the '@' symbol and the physical end-of-line marker is ignored.

- an opening round bracket, '(', implying that the line contains an *equation*.

The first non-space character may alse be an opening square bracket, '[', implying that the line begins with the name of a string or some other quantity which will be parsed according to the rules given in the next section. In this situation, the line after parsing must obey the rules given above.

Lines consisting only of spaces are ignored.

A *command line* should consist of a valid [B/D] command, followed by any pertinent arguments. The arguments must be separated from the command by at least one space, or by one colon. The command itself must consist of contiguous characters. Thereafter, as many spaces as desired can be strewn amongst the characters forming the arguments. For example, the following commands are all legal and equivalent:

BD>**adjust:[base1/base2]** ↩
BD> **adjust [base 1 / base 2]** ↩
BD>**adjust [b ase1/b ase 2]** ↩

1.4.2 Line parsing

Every line input to [B/D] is parsed before being passed to the command line interpreter. The parsing consists of replacing various of the characters in the line by other characters, and of the removal of redundant spaces. In particular we use the pair of square brackets '[ ]' to enclose text that might be replaced.

All command lines have leading and trailing spaces removed. All command lines ignore other spaces (except the space separating command from arguments) and are converted to lower case. Exceptions are the PRINT:, TITLE:, PRINT:, ITITLE:, and STRING: commands, where space and case conversions are not carried out except as necessary. Substrings are, however, parsed completely, for the STRING: command.

The parsing of lines to replace material enclosed in square brackets takes place before any other operation. The material that might be replaced is as follows:

- Control variables defined in active FOR: statements, such as $[i]$, which will be replaced by the current value of the control integer.

- Strings defined in a STRING: statement, such as $[a]$, which will be relaced by the current contents of the string called a.

- Integers defined in an equation of the form [(E)]. That is, the *equation* E is calculated and then rounded to give an integer which replaces all of [(E)]. The rounding convention is defined in Appendix A.

- Quantities of the form [<B:I>], where B is the name of a *base* and I is a positive integer. The effect is to replace $B : I$ by the name of the element being the Ith element of the base B, according to the *ordering convention*.

Priority is decided by deepest nesting. Unrecognised quantities in [. . .] are not erased from the command line.

1.4.3 Multiple lines

Multiple lines consist of distinct command lines separated by vertical line symbols, |. It is mostly irrelevant whether or not several commands appear alone on several lines or all on the same line separated by the | character. However, there are a few exceptions as follows.

- Where a multiple line follows a FOR: statement, all command lines included are executed accordingly, and no END: command is needed to terminate the loop.
- where a multiple line follows an IF: statement, all command lines included on the line are executed if the relevant condition is true.
- The following commands may not be used (as other than a first command) in multiple lines: REFRESH:; ENDIF:; FOR:; END:; IF:; REPEAT:; UNTIL:; WHILE:; WEND:.
- Macro-switching commands (RETURN:, GOTO:, and M:) are not allowed in multiple-line FOR: statements.
- an ELSE: command may be used in a multiple-line IF: statement, but not any other multiple-line command.
- if a macro-switch is found in a multiple-line IF: line, any succeeding commands are abandoned.

1.5 Line Continuations

Any line of input to BD which is terminated by linefeed/carriage return, and which has an ampersand character, & as its last non-space character means that the line is to continue to be read from the next line of input. If you are in interactive mode, you will receive a prompt, BD&, for the continuation. This continuation marker takes precedence over any parsing of the line, and is removed before parsing. Line continuations are allowed when inputting a string, a function, or a constant interactively. However, the possible length of a [B/D] *command line* is set at a maximum of 253 irrespective of whether or not the line is constructed from several physical lines.

1.5.1 Miscellaneous information

Numbers come in two breeds: standard, for reading in data and as parts of *equations*. The latter are not permitted to have exponent, whereas the former can. Thus the following is a legitimate standard number: `-14.06e-17`.

Output is flushed to output files (and to the *history file* and the *log file*, if these are in use) at the end of each command. If a crash happens during a command, any output from that command will be lost.

Errors are automatically echoed to screen - unless the job is a pure batch job, i.e. when channel 0 (the keyboard and screen) has never been referenced (read from or written to).

Chapter 2

Subroutines and external files

2.1 Overview

External files are associated with [B/D] by input and output channel numbers. Input channels link [B/D] to external files which contain [B/D] programs and/or subroutines. The links between channel numbers and external files are created by using the CHANNEL: command, and they may be removed by using the CLEARCH: command. The PATH: command may be used to allow the searching of other directories for any external files required. In cases where [B/D] is to share external files with other programs, such as text processors, the REFRESH: command should be used to update the external files.

Output channels offer destinations for output alternative to the screen, and are also used for the exportation of data and beliefs. Output is redirected by using the OCHANNEL: command, and data and beliefs may be exported using the ECHANNEL: command.

2.1.1 Subroutines and diversions

A subroutine, or macro, is simply a sequence of [B/D] commands. We might write subroutines to achieve some particular modularised task, or perhaps simply to break up a program into more manageable components. A number of commands are available to assist in this. In particular, the M: and RETURN: commands respectively direct program flow towards and away from a subroutine. The GOTO: command provides a simple means of jumping from one place in a program to another. Finally, the CLEARRETURNS: command can be used to override a sequence of nested return addresses from subroutines.

Currently, subroutines are not associated with direct facilities to vary parameters. However, it is possible to write subroutines which, by using *strings*, *constants*, and so forth, are reasonably abstract and capable of handling a variety of inputs. As an example, consider the fragment of code shown in Figure 2.1. The subroutine begins with a label, "@simplesub", which could be regarded as the name of the subroutine. The subroutine simply calculates the adjustment of one base by another, where the names of the two bases are assumed to be contained in the strings b and d, set outside the subroutine. Having performed the adjustment and output a result, the subroutine terminates. We will assume that these lines of code are contained somewhere in the external file called "sub.ex" in the current directory.

To use this subroutine, consider the fragment of code shown in Figure 2.2. This links the external file "sub.ex" containing the subroutine to input channel number 3. The following commands then use the subroutine twice, firstly achieving the adjustment of Ubase by Vbase and secondly the adjustment of Xbase by Ybase, where we assume these bases to be to have been defined earlier. Notice that we need not refer to channel 3 explicitly when we call the subroutine. The commands in Figure 2.2 might be issued interactively, or might themselves constitute part of an external file to which you have directed program flow.

2.1.2 Handling errors that arise during processing of a subroutine

The program can take two courses of action when an error occurs during a subroutine. Firstly, it can obey a set of commands that you specify in the case of error; and secondly it can take default action. The first case is described below: the ONERROR: command is used to set other than default action. The second case, the default action, is described here.

Figure 2.1: Writing a simple subroutine

@simplesub ↩
@ this subroutine calculates the adjustment of one base, whose name ↩
@ should be contained in the string b, by another base, whose name ↩
@ should be contained in the string d. ↩
print: Adjusting [b] by [d]. ↩
adjust: [[b]/[d]] ↩
print: The uncertainty resolved in the collection [b], having ↩
print: adjusted by the collection [d], is (rmtr). ↩
return: ↩

Figure 2.2: Using a simple subroutine

BD>**channel: i3=sub.ex ↩**
BD>**string: b=Ubase ↩**
BD>**string: d=Vbase ↩**
BD>**m: @simplesub ↩**
BD>**string: b=Xbase ↩**
BD>**string: d=Ybase ↩**
BD>**m: @simplesub ↩**

If an error occurs during a macro, and if no action has been specified via the ONERROR: command, the macro is aborted. If the error occurred during the initial execution of a log-in macro, the entire program is terminated. Otherwise, control returns to the screen.

A channel number need not be suplied with a label. If no channel number is given, the current channel number is assumed. If the named label does not exist on the current channel, a search is made until a matching file is found on another input channel, if any. These other channels are searched in numerical order.

2.2 Linking external files to [B/D]

▹ ▹ **Syntax**

1. BD>**channel: iI$_1$= [P] F ↩**

2. BD>**channel: oI$_2$= [P] F ↩**

where $1 \leq I_1 \leq 6$ *is an integer representing an input channel number;* $1 \leq I_2 \leq 4$ *is an integer representing an output channel number; F is a sequence of alphabetic characters representing a valid* filename*; and P is a sequence of alphabetic characters representing a path to a directory containing, or to contain, the file.*

◃ ◃

The CHANNEL: command is used to specify and attach external files, for output and input, to the program. The first form of the syntax is used to associate external files with input channels, of which there are six in all. The second form of the syntax is used to associate external files with output channels, of which there are four in all.

If no file path P is given, the sequence of directories given by the *search path* is searched in order to find the file F.

If an output channel is being declared, and if the external file cannot be found, then it will be created, either in the current directory or in the directory indicated by the path P if one was supplied. However, if an output channel is being declared and the external file does exist then its contents will be overwritten. Beware that you do not remove precious file contents accidentally.

If an input channel is being declared, and if the external file cannot be found, an error will be reported.

You may not associate an output channel with an external file which is currently associated with an input channel. You may not associate an input channel with an external file which is already associated with an input channel. You may associate an input channel with an external file that is already linked with an output channel (provided that the output channel is not currently being used for output; the attempt will result in the reporting of an error). In this case, the external file will be closed, and the output channel linked to it removed. Finally it is then linked to the input channel. This provides a limited means of interactively creating macro files as outputs for further use as inputs.

2.3 Removing links to external files

▹ ▹ **Syntax**

1. `BD>`**clearch: i**$\mathbf{I}_1$ ↩

2. `BD>`**clearch: o**$\mathbf{I}_2$ ↩

where $1 \leq I_1 \leq 6$ *is an integer representing an input channel number, and* $1 \leq I_2 \leq 4$ *is an integer representing an output channel number.*

◃ ◃

The CLEARCH: command is used to remove the link between an external input file and a channel number. Once the channel has been freed, it may be reallocated to another external file. The first form of the syntax is used to free input channels, and the second to free output channels. Optionally, the argument may be placed within parentheses.

2.4 Re-initialising macro handling

▹ ▹ **Syntax**

`BD>`**clearreturns:** ↩

◃ ◃

The CLEARRETURNS: command has no arguments. It is used to re-initialise the macro-handling system. That is, any outstanding return addresses from macro are removed, and the program is left awaiting keyboard input. This facility is chiefly useful after an error has occurred. Notice that the return address stack is automatically cleared when a RESTART: command is issued, but not when a CLEAR: command is issued.

2.5 Defining export channels

▹ ▹ **Syntax**

`BD>`**echannel: (I)** ↩

where $1 \leq I \leq 4$ *is an integer representing an output channel number.*

◃ ◃

The ECHANNEL: command selects the channel number for exported quantities. The channel number must previously have been associated with an external output file by using the CHANNEL: command.

This command only has any relevance for future EXPORT: commands. The export channel can be the same as the current output channel. In particular, material exported will be appended to this channel if it is already in use for the current [B/D] session.

2.6 Unconditionally switching program flow

▷ ▷ **Syntax**

1. `BD>`**goto: @L** ↩

2. `BD>`**goto: @L(C)** ↩

3. `BD>`**goto: (C)** ↩

where L *is the name of a* label *and* $0 \leq C \leq 6$ *is a valid* input channel number.

◁ ◁

The GOTO: command is used to switch program flow unconditionally to the address given. This command is thus the memoryless analogue of the M: command, and is not linked with a RETURN: command. You should be careful if you want to jump out of FOR: loops. The GOTO: command is not associated with a return address: therefore whenever the end-of-file marker is encountered after a GOTO: command has been issued, the program returns to the address associated with the most recent M: command or stops otherwise. The same holds when a RETURN: command is encountered after a GOTO: command. See the discussion in §2.7 for further details.

If the first form of the command is issued, with no channel number being specified, the channel number is deduced - [B/D] does not allow duplicate labels. In the third form of the syntax, the beginning of the file associated with the input channel is indicated. In common with other macro facilities, the channel number can be zero (meaning the keyboard): a label should not be supplied in this case. The address implied by the channel and label supplied must exist at the time the command is issued.

Commands which might result in diverting program flow (GOTO:, M:, and RETURN:) must be issued on a separate line; otherwise an error is reported. Hence the following usage is not permitted:

`BD>`**for: i=1,1,2 | goto: @test[i]** ↩

The following fragments of code show three examples.

`BD>`**goto: @lisbon** ↩
`BD>`**goto: @lisbon(2)** ↩
`BD>`**goto: (2)** ↩

The first example refers explicitly to the label "@lisbon". No channel number is given, so [B/D] will search all the input channels for a match. There may be at most one match as [B/D] does not permit duplicate labels. The address thus consists of the given label, on a channel which will be deduced. In the second example, the address consists explicitly of a label and a channel number. In the third example, no label is given, and the top of the external file associated with input channel number 2 will be assumed as the required address.

2.7 Temporarily diverting program flow to subroutines

▷ ▷ **Syntax**

1. `BD>`**m: @L** ↩

2. `BD>`**m: @L(C)** ↩

3. BD>**m: (C)** ↩

where L is the name of a label *and C is a valid* input channel number.

◁ ◁

The M: command is used to divert program flow to a subroutine. Typically, once the subroutine has been processed, program flow resumes at the line beyond the line containing the M: command (or control returns to the keyboard). Most subroutines are terminated by a RETURN: command, or by and end-of-file marker with similar consequences. This command is thus the similar to the GOTO: command except that it is linked with a RETURN: command.

If the first form of the command is issued, with no channel number being specified, the channel number is deduced - [B/D] does not allow duplicate labels. In the third form of the syntax, the beginning of the file associated with the input channel is indicated. In common with other macro facilities, the channel number can be zero (meaning the keyboard): a label should not be supplied in this case. The channel and label specified or implied must exist at the time the command is issued.

Technically, [B/D] maintains a stack of program diversions and return addresses. The return address is the line beyond the line containing the M: command if the command was issued from a macro file, or otherwise the return address amounts to returning control to the keyboard. The stack of return addresses may grow as large as machine memory will allow, meaning that you may nest subroutines as deeply as you wish. The RETURN: command acts as an end to the diversion or subroutine, and an end-of-file marker also acts in this way. In either of these circumstances, the return address stack diminishes one level, and control is passed back to the situation holding before the corresponding M: command was issued.

Although we tend to use the M: and RETURN: commands for the creation of separate program modules, this is not forced. You may use or misuse the M: command as you see fit. The CLEARRETURNS: command may be used to clear the return address stack if necessary. The return address stack is automatically cleared when a RESTART: command is issued, but not when a CLEAR: command is issued.

Commands which might result in diverting program flow (GOTO:, M:, and RETURN:) must be issued on a separate line; otherwise an error is reported. Hence the following usage is not permitted:

BD>**for: i=1,1,2 | m: @test[i]** ↩

The following fragments of code show three examples.

BD>**m: @oporto** ↩
BD>**m: @oporto(2)** ↩
BD>**m: (2)** ↩

The first example refers explicitly to the label "@oporto". No channel number is given, so [B/D] will search all the input channels for a match. There may be at most one match as [B/D] does not permit duplicate labels. The address thus consists of the given label, on a channel which will be deduced. In the second example, the address consists explicitly of a label and a channel number. In the third example, no label is given, and the top of the external file associated with input channel number 2 will be assumed as the required address. For each usage the return address will be the same, and as soon as the subroutine has been processed and a RETURN: command or end-of-file marker encountered, program flow will return to this point.

2.8 Redirecting output

▷ ▷ **Syntax**

BD>**ochannel: (I)** ↩

where either $I = 0$, representing the interactive screen, or $1 \leq I_2 \leq 4$ is an integer representing an output channel number.

◁ ◁

The OCHANNEL: command specifies the channel that should be used for output until a further OCHANNEL: command is issued. The output channel is reset to zero (interactive screen) whenever a RESTART: command is issued, but not when a CLEAR: command is issued.

As an example, consider the fragment of code shown in Figure 2.3. The commands in this fragment (1) link an output channel to an external file (which will be created if it does not already exist); (2) redirect output to this file; (3) print a message which will appear as text at the head of this file; and (4) redirect output back to the screen.

Figure 2.3: Redirecting output

BD>**channel: o3=newout.txt** ↩
BD>**ochannel: (3)** ↩
BD>**print: This message will appear on output channel 3.** ↩
BD>**ochannel: (0)** ↩

2.9 User-defined error handling

▹ ▹ **Syntax**

1. BD>**onerror: @L** ↩

2. BD>**onerror: @L(C)** ↩

3. BD>**onerror: (C)** ↩

4. BD>**onerror:** ↩

where L is the name of a label *and C is a valid* input channel number.

◃ ◃

The ONERROR: command is used to provide directions to a special error-handling macro, which should be found at the address specified by the label and/or channel supplied. If the first form of the command is issued, with no channel number being specified, the channel number is deduced - [B/D] does not allow duplicate labels. In the third form of the syntax, the beginning of the file associated with the input channel is indicated. In common with other macro facilities, the channel number can be zero, meaning the keyboard for input or the screen for output. In such cases, a label should not be supplied. The address specified or implied must exist at the time the command is issued.

If the fourth form of the syntax is used, any directions to an error-handling macro are removed, and the default status restored to no user-defined error handling. Otherwise, if this facility is activated and an error occurs, the facility is toggled off, to avoid infinite looping.

In the case of no ONERROR: direction being supplied and an error occurring, the macro being processed is aborted. If the error occurred during the initial execution of a log-in macro, the entire program is terminated. Otherwise, control returns to the screen.

In the case of alteration (either by re-definition of the channel, or by updating it via the REFRESH: command, or by clearing it) of the input channel containing the label and channel number specified in an ONERROR: command, the following steps are followed. If the channel no longer exists, or if it exists without the specified label, then it is as though the ONERROR: facility is switched off. Otherwise, the facility remains switched on. If an error occurs during the refresh process, the ONERROR: facility is switched off.

The effect of the ONERROR: command is to specify a subroutine which should be processed when an error occurs. After the subroutine has been processed, control should return to the line immediately following the line where the error occurred.

2.10 Defining additional search paths

▹ ▹ **Syntax**

BD>**path: directory1** [**: directory2**] [...] ↩

where directory1 and directory2 are the paths, in a format dependent upon the machine implementation, to the directories to be joined to the standard search path.

◃ ◃

The PATH: command is used to define additional search paths for input files. The effect is to add these paths temporarily (for the life of the session) to the search paths currently defined in the environment variable BDMACROS (see §1.3.2). The order of searching is always (1) the current directory first, (2) the directories given by any defined search paths: the most recently defined are searched first. Examples for UNIX and WINDOWS implementations are:

UNIX: BD>**path: bd/bdinputs/special/:bd/oldinputs** ↩

WINDOWS: BD>**path: bd\bdinputs\special\;bd\oldinputs** ↩

2.11 Updating shared external files

▹ ▹ **Syntax**

1. BD>**refresh: (C)** ↩

2. BD>**refresh:** ↩

where C is a valid input channel number.

◃ ◃

The REFRESH: command is used to update, or refresh, input channels. This will become necessary in the case of multitasking where one or more input channels currently announced to [B/D] via CHANNEL: commands are being updated by other programs. For example, you might be debugging a [B/D] macro interactively by running the macro, switching to a text editor to correct any mistakes, and then switching back to [B/D] to rerun the macro. [B/D] cannot automatically detect changes made to its input channels, and so needs informing that it should take account of such changes. Particular effects are to update the pointers to the labels on the input files, and to ensure that the ONERROR: facility will operate correctly. If an error occurs during the refresh process, the ONERROR: facility is switched off.

The first form of the syntax is used to refresh the single file corresponding to the specified channel number. The second form of the syntax is used to refresh all the external files currently corresponding to specified channel numbers.

2.12 Returning from subroutines

▹ ▹ **Syntax**

1. BD>**return:** ↩

2. BD>**return: E** ↩

where E is any equation.

◁ ◁

The RETURN: command is used to terminate the processing of a macro and return to the place immediately following the place where the last M: command was issued, or to the keyboard if the last M: command was issued interactively. You should regard M: and RETURN: commands as a matched pair during typical use. The second form of the syntax is used to make the return conditional on the value given by the accompanying *equation*. The equation is defined to be TRUE if the rounded value of the equation is equal to unity, and FALSE otherwise. If the evaluation of the equation causes an error, then the equation is taken to be true, and a return will be made unless superceded by an extant ONERROR: instruction.

Commands which might result in diverting program flow (GOTO:, M:, and RETURN:) must be issued on a separate line; otherwise an error is reported. Hence the following usage is not permitted:

BD>**for: i=1,1,2** | **return:** ↩

The return address stack is automatically cleared when a RESTART: command is issued, but not when a CLEAR: command is issued.

Chapter 3

Commands controlling program flow

3.1 Looping with a control variable

▷ ▷ **Syntax**

1. BD>**for: N1=E1,E2,E3** [**,N2=E4,E5,E6**] ... ↩

BD> ... ↩

BD> ... **Body of commands** ↩

BD> ... ↩

BD>**end:** ↩

2. BD>**for: N1=E1,E2,E3** [**,N2=E4,E5,E6**] ... | **Command** | ... | **Command** ↩

where $E1$, $E2$, ... , are any valid equations, and $N1$, $N2$, ... , are names, consisting of a sequence of alphanumeric characters beginning with an alphabetic character.

◁ ◁

The FOR: ... END: loops are used to provide loops in which a control integer varies, and where the control integer is accessible to the user. We consider two such kinds of loop. The first is our standard implementation, and the second is a more limited, but much faster, implementation designed to avoid some of the speed restrictions that arise from [B/D] being an interpreted language.

For both forms, the intention of the constructs are as follows. $N1, N2, \ldots$ are the names of control variables. The values of $E1, E2, E3, \ldots$ are rounded to the nearest integers, $I1, I2, I3, \ldots$, to give starting, stepping, and stopping values for the control variables. Thus, the intention of the loop with only the control variable $N1$ will be to repeat a block of commands with $N1 = I1$, $N1 = I1+I2$, and so forth. The stepping value $I2$ may be negative. In general, if the stepping value is positive then the loop continues whilst the control variable has a value not exceeding $I3$. If the stepping value is negative, then the loop continues whilst the control variable has value not less than $I3$. Furthermore, if the stepping value is zero; or if it is positive and the starting value exceeds the stopping value; or if it is negative and the starting value is less than the stopping value, then all the commands in the loop will be ignored.

For example,

BD>**for: i=1,-1,2** ... ↩

gives an instance of where the loop will be ignored as it is not possible to reach 2 from one by stepping negatively. On the other hand,

BD>**for: i=0, 2, 1, j=-1, 1, 0** ... ↩

establishes a loop which will work through a body of commands with the control variables taking these values in order: $(i = 0, j = -1)$ and $(i = 0, j = 0)$. Notice that the stepping value here for i implies the sequence $i = 0$, $i = 2$, $i = 4$, and so forth, but only the first value is encountered as the stopping value is $i = 1$.

Notice that entire command lines are parsed as one. Thus, the usage in the following example

BD>**for: i=1,1,2,j=1,2,[i], ...** ↩

will fail because at the time of parsing [i] at this point is unknown.

The first form of the syntax is our standard form whereby one line contains a FOR: statement (and nothing else); a number of commands on succeeding lines (which may contain lines with multiple commands); and finally an END: command to mark the end of the loop. The commands between the FOR: statement and the END: statement will be repeated as appropriate, according to the settings of the control variables. An example of the use of this form of the syntax is as follows.

BD>**for: i=1,1,2, j=1,1,2, k=1,1,2** ↩
BD>**print: [i][j][k]** ↩
BD>**end:** ↩

This will print out the following sequence of numbers: 111, 112, 121, 122, 211, 212, 221, 222; showing that the last control variable encountered in the statement, k, changes fastest.

The second form of the syntax is a special fast form, which we will call *the one-line FOR: statement*. It consists of the FOR: statement, together with one or more commands separated by the symbol ' | ', on the same line. However, the END: command is not (and may not) be used to mark the end of the loop: the end-of-line marker does this instead.

3.1.1 General remarks

1. To access the current value of the control variable whose name is $N1$, use the syntax '[$N1$]' as appropriate. (See the section on line parsing). The example just given illustrates this.

2. From the point of view of accessing the value of a control variable, if two or more control variables with the same name are active, then the most recently defined is used. Consequently, the action of the following (legal) sequence:

   BD>**for: i=1,1,2, i=4,5,9** ↩

   BD>**print: [i]** ↩

   BD>**end:** ↩

   would be to print out the values 4 and 9 twice.

   Furthermore, the names of control variables used for FOR: loops take precedence over strings with the same name.

3. For interactive work, when you issue a standard FOR: command, the program simply copies your further inputs (without obeying any commands) until you input an END: command to mark the end of the loop. Thereafter the loop is processed in the usual way. The interactive FOR:-END: construction causes the copying of command lines to a temporary file. Every line is copied as an original, without any parsing: errors in the command syntax may be reported, but the process will continue. During such interactive work, the following prompt is used:

   BD/ ↩

   to indicate to you that the program is awaiting an END: statement before processing your inputs.

4. There are no limits to the number of active FOR: loops, subject to memory limitations.

5. The number of control variables that can be defined in one FOR: statement is limited to 30.

6. All commands are permitted during the processing of a standard for–end loop. However, certain commands are disallowed during processing of a one-line FOR: statement. These include

   (a) channel switching via M:, GOTO:, and RETURN: commands;

   (b) if-then-else constructions via IF:, ELSE:, and ENDIF: commands;

   (c) other looping via FOR:, END:, REPEAT:, UNTIL:, WHILE:, and WEND: commands;

   (d) channel closing via the REFRESH: command.

7. The names and details of all active FOR: statements can be checked by issuing a LOOK: command with the for argument. Control variables defined in the same FOR: statement are marked “not final” except for the first.

3.2 Conditional statements

▷ ▷ **Syntax**

1. BD>**if: E** ↩

BD> . . . ↩

BD> . . . **Body of commands** ↩

BD> . . . ↩

BD>**endif:** ↩

2. BD>**if: E** ↩

BD> . . . ↩

BD> . . . **Body of commands** ↩

BD> . . . ↩

BD>**else:** ↩

BD> . . . ↩

BD> . . . **Body of commands** ↩

BD> . . . ↩

BD>**endif:** ↩

3. BD>**if: E** | **command** | **command** | . . . ↩

4. BD>**if: E** | **command** | **command** | . . . | **else:** | **command** | **command** | . . . ↩

E is any valid equation.

◁ ◁

These constructions are used to make program flow conditional on some aspect defined by the user. The value of the equation E is used to decide upon the appropriate course of action. In the context of if–then–else constructions, an equation is defined to be TRUE if the rounded value of the equation is equal to unity, and FALSE otherwise (the kind of rounding is defined in Appendix A). Therefore, if the value of the equation following the IF: command is equal to unity, the body of commands immediately following will be processed.

The first and second forms of the syntax are our standard implementations, and the remaining two are more limited, but rather simpler, implementations designed to avoid some of the problems that arise from [B/D] being an interpreted language.

If the first form of the syntax is used, if the equation is TRUE, then the body of commands up to the ENDIF: command is processed in the usual way. If, however, the equation is false, none of the commands between the IF: and ENDIF: are processed. In either case, the ENDIF: command terminates the construction.

If the second form of the syntax is used, if the equation is TRUE, then the body of commands up to the ENDIF: command is processed in the usual way. If, however, the equation is false, none of the commands between the IF: and ENDIF: are processed and instead the body of commands between the ELSE: and ENDIF: commands are processed. In either case, the ENDIF: command terminates the construction.

The final two forms of the syntax are special forms, which we will call *one-line* IF: *statements*. They consist of the IF: statement, together with one or more commands separated by the symbol ‘|’, on the same line. The fourth form shows that the ELSE: command may be included to provide an alternative set of commands to process if the condition turns out to be FALSE. Notice that the ENDIF: command is not (and may not) used to mark the end of the loop: the end-of-line marker does this instead.

3.2.1 Other remarks

1. If an error occurs in the definition of the equation used to generate the condition, the default action is for the value of the equation to be taken as FALSE.
2. All commands are permitted during the processing of a standard if–then–else construction. However, certain commands are disallowed during processing of a one-line IF: statement. These include
   (a) other if-then-else constructions via the IF:, ELSE:, and ENDIF: commands;
   (b) looping via FOR:, END:, REPEAT:, UNTIL:, WHILE:, and WEND: commands;
   (c) channel closing via the REFRESH: command.
3. There are no limits to the number of active if ... else ... endif loops, subject to memory limitations.
4. It is permitted to change channels, and so forth, via the M:, GOTO:, and RETURN: commands.
5. The current status of any if–then–else constructions are cleared whenever the RESTART: command is issued, but not when a CLEAR: command is issued.
6. For interactive work, when you issue a standard IF: command, the program simply copies your further inputs (without obeying any commands) until you input an ENDIF: command to mark the end of the construction. Thereafter the commands are processed in the usual way. The interactive construction causes the copying of command lines to a temporary file. Every line is copied as an original, without any parsing: errors in the command syntax may be reported, but the process will continue. During such interactive work, the following prompt is used:

   BD< ↩

   to indicate to you that the program is awaiting an ENDIF: statement before processing your inputs.

3.3 Looping until a condition is satisfied

▷ ▷ **Syntax**

BD>**repeat:** ↩
BD> ... ↩
BD> ... **Body of commands** ↩
BD> ... ↩
BD>**until: E** ↩

where E is any valid equation.

◁ ◁

The REPEAT: ... UNTIL: construction is used to provide a loop surrounding a body of commands which will be processed at least once, and will continue to be processed the equation E becomes TRUE. The equation is defined to be TRUE if the rounded value of the equation is equal to unity, and FALSE otherwise. If no equation is given, or if a mistake is made in determining the equation, then [B/D] assumes that a valid equation with value TRUE has been supplied, and the loop will not be repeated.

Both the initiating REPEAT: and the terminating UNTIL: command must be given on a separate line, and you should be careful not to confuse separate repeat ... until structures where you are nesting them.

As an example, consider the following program fragment.

BD>**c: %i=0** ↩
BD>**repeat:** ↩
BD> **c: %i=%i+1** ↩
BD> **c: %j=0** ↩
BD> **repeat:** ↩
BD> **c: %j=%j+1** ↩
BD> **print: (%i), (%j)** ↩
BD> **until: %j=2** ↩

BD>**until: %i=2** ↩

This code establishes two loops during which the constants take the values (%i=1,%j=1), (%i=1,%j=2), (%i=2,%j=1), (%i=2,%j=2), and then terminate.

3.3.1 Other remarks

1. The REPEAT: and UNTIL: commands may not be used during one–line FOR: and one–line IF: statements.
2. The while ... wend construction is similar to the repeat ... until construction, except that the test for processing the enclosed body of commands is at the start rather than the end of the loop. Therefore the body of commands enclosed in a while ... wend loop might not be processed, whereas the body of commands enclosed in a repeat ... until loop are always processed at least once.
3. There are no limits to the number of active repeat ... until loops, subject to memory limitations.
4. The current status of repeat ... until constructions are cleared whenever the RESTART: command is issued, but not when a CLEAR: command is issued.
5. For interactive work, when you issue a REPEAT: command, the program simply copies your further inputs (without obeying any commands) until you input an UNTIL: command to mark the end of the loop. Thereafter the loop is processed in the usual way. The interactive REPEAT:-UNTIL: construction causes the copying of command lines to a temporary file. Every line is copied as an original, without any parsing: errors in the command syntax may be reported, but the process will continue. During such interactive work, the following prompt is used:

   BD/ ↩

   to indicate to you that the program is awaiting an UNTIL: statement before processing your inputs.

3.4 Looping while a condition is satisfied

▷ ▷ **Syntax**

BD>**while: E** ↩
BD> ... ↩
BD> ... **Body of commands** ↩
BD> ... ↩
BD>**wend:** ↩

where E is any valid equation.

◁ ◁

The WHILE: ... WEND: construction is used to provide a loop surrounding a body of commands which will be processed as long as the rounded value of the equation E remains TRUE. An equation is defined to be TRUE if the rounded value of the equation is equal to unity, and FALSE otherwise. If no equation is given, or if a mistake is made in determining the equation, then [B/D] assumes that a valid equation with value FALSE has been supplied, so that the commands in the loop will not be processed.

Both the initiating WHILE: and the terminating WEND: command must be given on a separate line, and you should be careful not to confuse separate while ... end structures where you are nesting them.

As an example, consider the following program fragment.

BD>**c: %i=0** ↩
BD>**while: %i<2** ↩
BD> **c: %i=%i+1** ↩
BD> **c: %j=0** ↩
BD> **repeat:** ↩
BD> **c: %j=%j+1** ↩
BD> **print: (%i), (%j)** ↩

BD> **until: %j=2** ↩
BD>**wend: %i=2** ↩

This code establishes two loops during which the constants take the values (%i=1,%j=1), (%i=1,%j=2), (%i=2,%j=1), (%i=2,%j=2), and then terminate.

3.4.1 Other remarks

1. The WHILE: and WEND: commands may not be used during one–line FOR: and one–line IF: statements.
2. The while ... wend construction is similar to the repeat ... until construction, except that the test for processing the enclosed body of commands is at the start rather than the end of the loop. Therefore the body of commands enclosed in a while ... wend loop might not be processed, whereas the body of commands enclosed in a repeat ... until loop are always processed at least once.
3. The current status of while ... wend constructions are cleared whenever the RESTART: command is issued, but not when a CLEAR: command is issued.
4. There are no limits to the number of active while ... wend loops, subject to memory limitations.
5. For interactive work, when you issue a WHILE: command, the program simply copies your further inputs (without obeying any commands) until you input a WEND: command to mark the end of the loop. Thereafter the loop is processed in the usual way. The interactive WHILE:-WEND: construction causes the copying of command lines to a temporary file. Every line is copied as an original, without any parsing: errors in the command syntax may be reported, but the process will continue. During such interactive work, the following prompt is used:

   BD/ ↩

   to indicate to you that the program is awaiting a WEND: statement before processing your inputs.

Chapter 4

Strings

4.1 Defining strings

▹ ▹ **Syntax**

BD>**string: S=N** ↩

where S is the name of a string*, and N is the string.*

◃ ◃

Strings are sequences of alphanumeric characters of a length varying between zero and currently 253 characters. It is possible to define strings having length zero, which we term empty strings. Strings are mostly used (1) to define quantities that might take different values for different parts of a [B/D] program; (2) as shorthand for some long sequence of characters. Some example definitions are

BD>**string: a=xvariable1** ↩
BD>**string: b2=intercept** ↩

Commonly, command lines have leading and trailing spaces removed, and then other spaces (except the space separating command from arguments) are ignored and remaining characters converted to lower case. However, the STRING: command is an exception: space and case conversions carried out only as necessary. This allows the definition part to be used for titles, further commands, and so forth, where the retention of case or spacing might be crucial: when the string is actually used, parsing will be carried out in the usual way.

The names of strings *are* subjected to space and case conversions, so that the two string names 'TEXT1' and 'text1' are equivalent. Substrings (see below) are, however, parsed completely. A string defined with the same name as a pre-existing string simply overwrites the former definition. String definitions can be checked by issuing the LOOK: command with argument s. (When you do this, notice that a good deal of [B/D] output has names printed with the first character capitalised - this is cosmetic only.)

4.2 Using strings

4.2.1 Inserting strings

Wherever the name of a string appears in square brackets during [B/D] input, the name of the string is replaced by the string itself. Thus, for example, suppose that we have defined the strings $a = xvariable1$ and $b2 = intercept$ as above. Then the command

BD>**print: We are about to print the value of** [**a**] **and** [**b2**] ↩

is *parsed*, and the string names replaced, to obtain the command

BD>**print: We are about to print the value of xvariable1 and intercept** ↩

Unrecognised quantities in square brackets are not replaced. This thus allows the recursive definition of strings as in the sequence of commands

BD>**string: a=[b][c]** ↩
BD>**string: b=text1** ↩
BD>**string: c=text2** ↩

which would result in a later use of [a] being replaced by the characters 'text1text2'. Another use of this feature allows us to store entire command lines as strings. Consider, for example, the following definition:

BD>**string:t=print: This is an example!** ↩

At a later stage we might issue simply the 'command'

BD>[**t**] ↩

which, after parsing, will become equivalent to

BD>**print: This is an example!** ↩

Strings may be nested to any depth, limited only by line length. As an example, consider the following code:

BD>**string: mouse=cat** ↩
BD>**string: cat=dog** ↩
BD>**print:** [[**mouse**]] ↩

This fragment of code should print the text 'dog' as $[[mouse]] = [cat] = dog$.

In the context of inserting strings during the parsing of command lines, strings have lower precedence than all control variable names defined in active FOR: statements. Thus the following legal sequence of commands:

BD>**for: i=1,1,2** ↩
BD>**string: i=7** ↩
BD>**print: i=**[**i**] ↩
BD>**end:** ↩

results in the output 'i=1' and 'i=2', as the quantity '[i]' is replaced by the value of the control variable rather than the string.

See §1.4.2 for futher details on parsing.

4.2.2 Substrings

We implement substrings of strings in the following way. Consider the following sequence of commands:

BD>**string: A=abcdefghi** ↩
BD>**string: B=tuv[A \ 3,6]wxyz** ↩

The first command defines a string whose name is 'A'. The second command defines a second string, called 'B', whose definition part includes a substring extracted from the string whose name is 'A'. Indeed, the syntax '$[A\backslash 3,6]$' means that we should replace '$[A\backslash 3,6]$' by the third, fourth, fifth, and sixth characters of the string 'A'. Thus, 'B' will be defined as 'tuvcdefwxyz'.

In general, if $[name\backslash a,b]$ is detected as input, then 'name' is assumed to be the name of a string and a and b to be integers (or equations which will be rounded to integers) marking the desired substring of this named string. The a^{th} to the b^{th} characters inclusive of this named string then replace the piece of input in square brackets. The square brackets are also removed.

The rules governing substrings are as follows. Suppose that we encounter the sequence of characters $[N\backslash a,b]$ in a command line. N should be the name of a string and a and b are equations which will be rounded to the nearest integer. Suppose that L is the length of the string whose name is N.

- $[N\backslash a,b]$ is the substring formed by the characters $a \ldots b$ of N whenever $1 \leq a \leq b \leq L$.
- if $a = 0$ then $a = 1$ is assumed, and a warning is given.
- if $b > L$ then $b = L$ is assumed, and a warning is given.
- if $a < 0$ then $a = L - b + 1$. This gives us a way of referring to the final b characters of N.
- if $a > b$ then the substring is empty, so that the construction $[N\backslash a,b]$ is simply removed.

4.3 Testing strings

The following string operators, discussed in greater detail in §21.11, may be used in equations:

empty$() to test whether or not a string is empty.

eq$(,) to test the equality of two strings.

len$() to return the length of a string.

neq$(,) to test the inequality of two strings.

real$() to test whether a string represents in integer; a real number; or neither of these.

These string operators all return the value unity if the test is true, and zero otherwise, and all may be used in equations in the usual way.

4.4 Inputting strings interactively

▷ ▷ **Syntax**

`BD>`**readstring: N=** ↩
`BD-` ↩

where N is the name of a string*, and S is the string.*

◁ ◁

We use the <u>READSTRING:</u> command to prompt for interactive input of a string. The command typically will be issued from a file being run as as part of a macro. When it is issue, the prompt

`BD-` ↩

will appear on the screen, and you should type in the string S, followed by the 'Enter' key to terminate it. The string so defined is then available for use in exactly the same way as strings defined via the more usual <u>STRING:</u> command. If you press the 'Enter' key immediately on receiving the prompt, the string will be defined as empty.

4.5 Deleting strings

▷ ▷ **Syntax**

`BD>`**xstring:** S_1 [, S_2, ...] ↩

where $S_1, \ldots$ *are the names of* strings.

◁ ◁

Strings are deleted using the <u>XSTRING:</u> command, which takes as its arguments the names of strings to be deleted. It is not an error if the name supplied is not currently recognised as the name of a string. Wildcards may be used in the usual way to delete many strings simultaneously.

Chapter 5

The command history

5.1 Introduction

Each line of input that you type in from the keyboard is retained in a a list called the command history. The following rules apply to what is retained.

1. Line continuations (using the & character) are interpreted in the natural way: that is, the concatenation of the appropriate text is stored as a single line in the command history.

2. The input line is retained without any parsing whatsoever, so that input lines which include features such as [...] which might ordinarily be replaced by other text (see §1.4.2) retain these features unparsed.

3. The maximum number of input lines retained depends upon a fixed number (set at compile time) whose value can be seen *inter alia* by issuing the LOOK: command with the program argument. Whenever the command history becomes full, the older half is removed.

4. Each input line retained in the command history has a unique address which we can refer to for editing.

5. Empty lines are not retained in the command history.

6. There are two particular commands, the !: and /: commands, which act only on the lines retained in the command history. The former views the command history and replays old command lines as fresh input, and the latter removes input lines from the command history. The input lines containing these commands and their arguments are never retained in the command history.

7. The input lines which form the command history can be retained optionally on a file which we call the history file, which is described and set up using the history argument to the KEEP: command.

8. The RESTART: command empties the command history, and flushes it to the history file if necessary. The CLEAR: command leaves alone both the command history and the history file.

The two commands used with regard to the command history are special, and treated unlike standard BD commands. They consist of the !: and /: commands, which are used for replaying, viewing and deleting keyboard input. These commands are issued like ordinary commands, but are extraordinary because they must appear as described, without assuming parsing of the input line (their arguments, however, are subject to parsing in the usual way). This means that you may not use one of these two commands as the result of a string. An additional feature is that they do not themselves appear as items in the command history. Unlike most other [B/D] commands, these commands are not renameable. Each command can be followed optionally by a colon or a space (as for standard commands) but need not be. Furthermore, these commands must be issued as the first non-space character on a line: for example, you may not issue such a command as the second command of a multiple line.

Neither of these commands ever appears in the command history (see below.) However, the !: command can result in the replay of a previous line input from the keyboard which will be added anew to the command history.

5.2 Viewing and replaying the command history

▷ ▷ **Syntax**

1. BD>**!** ↩

2. BD>**!:E** ↩

where E is any valid equation *which will be rounded to an integer.*

◁ ◁

The !: command is used to review the command history, and possibly to replay one of the input lines contained therein. The first form of the syntax simply displays the command history. Each line of keyboard input is temporarily retained in this command history with a unique address, an integer which is also displayed. Notice that this command history will remain unaffected by issuing this command.

The second form of the syntax calls for one of the lines retained in the command history to be recalled and replayed. There are two possible ways of recalling input lines: by absolute and relative addressing. For either usage, the equation E is parsed in the normal way before the value of the equation is determined. If the actual or relative address indicated by the value of E does not match an entry in the current command history then no action is taken. Notice that the replacement takes place even within an inactive if-then-else construction, and before the command history is updated.

For absolute addressing, the !: command is issued with an *equation* which will be rounded to an integer. The integer refers to the unique address of a retained input line. For this usage of the !: command, the input line containing the !: command will not appear in the command history, but the input line to be replayed will be retained anew.

For the relative addressing usage, the command is issued with a negative value, so that the rounded value of E should be negative. These circumstances mean that the input line indicated by going back $\mid E \mid$ places in the current command history is to be replayed as the current line. As before, the input line containing the !: command will not appear in the command history, but the replacement will.

5.3 Deleting items from the command history

▷ ▷ **Syntax**

BD>/ **E1, E2,** ... ↩

where $E1, E2, \ldots$, are equations *which will be rounded to integers.*

◁ ◁

The /: command is used to delete input lines from the command history so that they are not saved to a history file. This would be appropriate to remove input lines containing mistakes, etc., from the command history. The equations $E1, E2, \ldots$ are parsed in the normal way before the value of the equation is determined.

The rounded value of each equation is intended to represent the integer address of an item retained in the command history. If this address does not match any entry in the current command history then no action is taken. Otherwise, the effect is to remove from the command history the input lines whose addresses are the rounded values of $E1, E2, \ldots$. (The addresses can be reviewed beforehand by issuing the !: command.) If no arguments are given, then nothing happens. Whichever, this command line is not reproduced in the command history.

Chapter 6

Generating beliefs

6.1 Introduction

There are three types of belief-carrying quantity. They are elements, components, and assignments, each discussed further below. There also *data-carriers*, which are quantities that have been defined by DATA:, DATAVEC:, and FACTOR: commands, but which are not associated with any belief specifications. Their sole function is to carry data. As soon as an element with the same name is introduced, the data is attached to the element, and the distinction removed.

6.2 Abbreviations for constructing collections

The following remarks apply to some of the commands which need to generate a collection in the definition part. In particular, these commands are XBASE:, XDATA:, XGRID:, SUMMARY:, BASE:, XELEMENT:, COHERENCE:, SHOW:, CENTRE:, EXPORT:, and LOOK:.

6.2.1 Wildcards

Two wildcard symbols are available for use in the [B/D] commands listed below. The two wildcard characters are the question mark, ?, and the dollar symbol, $. They are used respectively to match any individual character and either any terminating sequence of characters or any beginning sequence of characters. The two wildcards operate independently, and can be used simultaneously.

**$** is used at the beginning or end of any string to match any sequence of characters and no characters. For example, suppose that we consider only the four strings x.1.17, x.1.2, zy.1.2, and yy.3.2. Then x$ refers to the pair x.1.17 and x.1.2, and $2 to the pair x.1.2 and yy.3.2. The $ symbol on its own refers to all possible strings, and this is useful for global declarations. Note that both $x.1.2 and x.1.2$ match x.1.2.

? is used to match any single character in a string. Thus ?y.?.2 refers to the pair zy.1.2 and yy.3.2.

Examples are as follows:

1. The following instruction deletes all elements such as x11, x21, x11.7, etc., whose name consists of at least three characters beginning with x and with third character 1.

   `BD>`**xelement:x?1$** ↩

2. The following command deletes all elements.

   `BD>`**xelement:$** ↩

3. The following command deletes all elements whose names consist of four characters.

   `BD>`**xelement: ????** ↩

The wildcard facilities are restricted to the commands listed at the beginning of the section, for elements and bases, and also to XASSIGN: for deleting assignments, XSTRING: for deleting strings, XC: for deleting constants, and XF: for deleting functions.

6.2.2 Abbreviated notation for bases and elements

It is often useful to be able to use abbreviations for collections to be used in constructing other collections. In particular, we may like to use a *base* name that we supply as an argument to some of the commands listed at the beginning of the section in different ways. To do this we introduce a particular notation, which applies to the bases named in the definition part of the ADJUST:, TESTGRID:, BASE:, XBASE:, XGRID:, and LOOK: (with argument b) commands. (For the ADJUST: command, see also §9.1.2.)

For these commands, a base D can appear as D, $< D >$, or $\{D\}$, with the following interpretation.

D: This is the simplest usage, and means the base D, is added as itself.

$< D >$: This means that the base D is replaced by its constituent bases and elements, in alphabetical order. The notation applies only to a single base, so the notation $< D, E >$ is not allowed. As an example, if $D = [X2, C1, Z2, X1]$ where $X1, X2$ are elements and $C1, Z2$ are bases, then

$$F, < D >, G \Longrightarrow F, C1, X1, X2, Z2, G.$$

$\{D\}$: This means that the base D is replaced by its constituent *elements* in alphabetical order. Any bases contained in D are likewise replaced by their consituents and so forth, recursively, until only a collection of elements is left. The notation applies only to a single base, so the notation $\{D, E\}$ is not allowed. As an example, suppose that $D = [X2, C1, Z2, X1]$ where $X1, X2$ are elements and $C1, Z2$ are bases, where $C1$ contains the elements $A1, A2$ and $Z2$ contains the element $A0$. Then

$$F, \{D\}, G \Longrightarrow F, A0, A1, A2, X1, X2, G.$$

For the remaining commands listed at the beginning of this section, that is the XDATA:, XELEMENT:, SHOW:, COHERENCE:, SUMMARY:, EXPORT:, and CENTRE:, commands, and the remaining usages for the LOOK: command, any use of abbreviation notation is ignored. Each of these commands requires eventually a simple list of elements, so any base names given in their definition parts are simply replaced by their constituent parts, recursively replacing all constituent bases as appropriate.

6.2.3 Base exclusion

For all of the commands listed at the beginning of the section, we are able to specifically exclude some collections. This is achieved by marking a base or element name with the ^ symbol as a suffix. The usage is best illustrated by example. Suppose, for example, that we have defined the bases $B = \{V, C, Z\}$ and $C = \{X, Z\}$. Then we could define the collection E as in

$$E = \{B, C^\wedge\} \Longrightarrow E = \{V, Z\}.$$

That is, the base E is defined to be B and not C, where the base C is excluded. This takes place sequentially, so that for example,

$$E = \{B, C^\wedge, X\} \Longrightarrow E = \{V, X, Z\}.$$

We can use this facility in conjunction with abbreviation notation as described in §6.2.2. For example,

$$E = \{B, \{C^\wedge\}\} \Longrightarrow E = \{V\},$$

where the constituents of C, rather than C itself, are removed.

6.2.4 Abbreviating collections of data-carrying elements

We may use an abbreviation to help us to construct bases quickly. Suppose that B is any base. Then $B+$ refers to all the elements in the base B possessing data; and the symbol '+' alone signifies all elements defined so far and possessing data. This facility is available for all of the commands listed at the beginning of the section. Note that when the form $B+$ is used, only the data-carrying constituents of the base are included. If B contains other bases, they are not included or searched in any way.

6.3 Functional forms and indices

These remarks about declaring functional indices apply to the following commands: the creation of functions via the F: command, the declaration of assignments via the ASSIGN: command, and the specification of functional beliefs over components and between components and elements via the FVAR: and FE: commands.

The quantities addressed by these commands can be defined so that they have indices that are allowed to vary and will later take *positive* integer values (assignments are a special exception). Suppose, for example, that you wished to generate an ANOVA type model of the form $Y_{ij} = m + a_i + b_j + e_{ij}$. We could explicitly create as elements all the terms given here, but it is more convenient and natural to define them as one assignment with indices that are permitted to vary as necessary. For example we might establish this model in [B/D] by issuing the command

BD>**assign Y.i.j=m+a.i+b.j+e.i.j** ↩

Notice that we use the full stop character '.', followed by an alphabetic character to represent an index that can vary. When integer values to replace the indices become known, the integers replace the alphabetic characters, but not the full stop. Thus, were we actually to create an element from the assignment with $i = 2$ and $j = 3$, we would create an element with the name 'Y.2.3'. Moreover, the act of creation would make reference to the quantities 'a.2', 'b.3', and 'e.2.3'. The alphabetic characters representing a particular index must be unique to the occasion. That is, it is not permitted to define the assignment $Y.i.i$.

Given a particular command, we call the part to the left hand side of the equals symbol the *naming* part, and the part to the right hand side the *definition* part of the construction. Thus here $Y.i.j$ is the naming part, and $m + a.i + b.j + e.i.j$ is the definition part of the construction.

To avoid confusion about when an index is not an index, we insist that the *naming part* may not contain both varying indices and integer indices. That is, the assignment name 'X.2.i' is not allowed as it contains both integer and varying indices. However, as we have seen 'Y.i.j' is allowed as it contains only varying indices, and 'Z.4.1' is allowed as it contains only integer indices. However, the *definition part* is not limited in this way, so that you may mix varying and integer indices freely. For example, the following modification to the previous example is permitted:

BD>**assign Y.i.j=m+a.i+b.j+e.2.j** ↩

6.3.1 Modified indices

In the *definition part*, but not the *naming part*, varying indices may be modified by the addition, subtraction, multiplication or integer division of or by an integer. There form for modifying indices is as shown in the following example:

BD>**assign A.i.j = B.(i+1).(j-1) + C.(i*5).(j/2)** ↩

For example, the specification of $i = 1, j = 4$, implies the assignment $A.1.4 = B.2.3 + C.5.2$.

6.4 Declaring and using linear combinations

6.4.1 Overview

[B/D] allows the storage of linear relationships, which we call *Assignments*. These do not carry beliefs per se, but play a key role in creating elements from the linear relationships so stored. Typical assignments consist of a linear combination of *elements* and *components* and may include a constant part. Often we define assignments to have one or more indices which might vary.

The **ascf** and **ascl** operators can be used to inspect the coefficients of the components of assignments. These operators take into account any varying indices.

The ASSIGN: command is used to define the linear relationships, and the XASSIGN: command is used to delete them. The assignments are useful (1) for the generation and regeneration of beliefs from specified linear combinations, (2) for the efficient storage of many linear combinations by virtue of their ability to handle functional beliefs and so forth. The LOOK:(a) command can be used to inspect some or all of the assignments defined.

6.4.2 Declaring assignments

▷ ▷ **Syntax**

BD>**assign** A [*Indexlist*] $= [(E_1)][N_1\ ]\ [\ \pm\ [(E_2)][N_2\ ]\ [\ \pm \ldots]] \hookleftarrow$

where A *is an alphanumeric representing the name of an* assignment, $N_1, \ldots$ *are names of quantities, Indexlist is a list of varying indices, and* $E_1, E_2, \ldots$ *are* equations.

◁ ◁

The ASSIGN: command is used to establish linear relationships. The naming part of the ASSIGN: command consists of a name A followed optionally by a list of varying indices. The name A might include integer indices, but as it is not possible to mix integer and varying indices in the naming part, this would preclude the option of following the name with a list of varing indices.

Indexlist is a list of varying indices, i.e. a period symbol '.' followed by an alphabetic character. Where there are many indices, a different alphabetic character must be selected for the different indices.

$E_1, E_2, \ldots$ are valid *equations* which will serve as coefficients for the quantity which it precedes, if any. Each equation may refer to the - as yet unknown - indices given in the naming part. The equations are not evaluated until the assignment is accessed, so these equations may also contain functions and other material whose values are as yet unknown or undefined. It is not necessary for an equation to act as a coefficient, as these unaccompanied equations are treated as scalar parts to the assignment.

Each of $N_1, N_2, \ldots$ is typically one of the following:

- an element or base name
- an assignment name
- a component name

You are not restricted to these choices, but they are the only meaningful types from the point of view of building the assignment: unrecognised quantities are ignored. (This may mean that a spelling mistake passes unnoticed.) As for the equations serving as their coefficients, these names are accessed only when the assignment is actually used, and so they need not exist when the ASSIGN: command is given, but should exist when the assignment is used.

6.4.3 Examples

In the first example, the assignment named A is defined to be the linear combination $B + 2C + E.1$. None of these quantities need exist at this point, and if they do not exist when the assignment is used, they will be ignored. The examples include such quantities as %c (meaning the *constant* c) and #kda.s (meaning the *function* kda.s), which are described more fully in chapter 7.

1. BD>**assign: A=B+(2) C+E.1** ↩
2. BD>**assign: G.3.4=(%c)+H.2+F+ (26)** ↩
3. BD>**assign: D.s.t=(#kda.s+#kdb.s) D.s.(t-1) + (#kdb.s) D.s.(t-2)+(#kdc.s)+E.s.t** ↩
4. BD>**assign: F.s.t=B.s.t+B.1.1** ↩

In the second of these examples, the assignment named $G.3.4$ is defined to be the linear combination $H.2 + F + 26 + \%c$, where %c is some constant which will be evaluated at this point (use a function rather than a constant if you wish to delay the evaluation). Notice that this example contains two scalar parts.

The third example is taken from a genuine application, and involves part of the specification of an autoregressive error structure. It makes the definition

$$D_{s,t} = (k_s^{(1)} + k_s^{(2)})D_{s,t-1} + k_s^{(2)}D_{s,t-2} + k_s^{(3)} + \epsilon_{s,t}$$

where the k's are functions of s, the D's are vectors of quantities related to similar previous quantities via the given relationship, and the ϵ terms are noises.

The fourth example shows a definition part which consists of a component which includes varying indices, and a component which does not. Notice that this results for example in the definition of $F.1.1 = B.1.1 + B.1.1$, so that the component $B.1.1$ is repeated. This is permissible, and the definition part will be taken to be $(2)B.1.1$.

6.4.4 Using assignments

Assigned quantities are generally used for constructing beliefs. Beliefs about themselves may be constructed by a subsequent BUILD: command, or they may be included in the definition part of other BUILD: commands, without being themselves constructed. Assignments defined with varying indices are particular useful for constructing many related quantities. Refer to the BUILD: and COBUILD: commands for further details.

Beware of making circular specifications, for example:

BD>**assign: X=Y** ↩
BD>**assign: Y=X** ↩

This is a permitted sequence of definitions, but the attempt to construct X would fail as the sequence of definitions is circular, and [B/D] would report an error.

If there is any confusion between assignment and element names in the definition part of an ASSIGN:, BUILD: or COBUILD: command, because an assignment of the same name as an element exists, the element name takes priority. The assignment name can be given priority by explicity prefixing it with the ! symbol, as in the example:

BD>**assign: X.2=!Y.1** ↩

which defines the assignment $X.2$ to be explicitly the *assignment* whose name is $Y.1$ even if an *element* with the same name exists.

Assigned elements are accessed in such a way that cross-products and covariances are correcly evaluated for both the BUILD: and COBUILD: commands, so as correctly to determine the variance of the quantity that you are building.

Various commands allow the automatic creation of assignments where the linear combination is a more natural quantity to be retained than beliefs about them. As an example, the canonical quantities that result from an adjustment can be retained as assignments. The **ascf** and **ascl** operators can be used to inspect these assignments.

6.4.5 Recursion in assignments

Recursion is available for assignments. For example, suppose that you

wish to generate a random walk with noise: $X_t = X_{t-1} + V_t$. This could be generated via the following assignment:

BD>**assign X.t=X.(t-1)+V.t** ↩

Whenever we reference this assignment, [B/D] notes the recursive aspect and deduces, by back tracking through the recursion, what is actually intended. This requires some termination criterion. By default, any quantity with a varying index which becomes zero, becomes zero itself. (If a varying index becomes negative, this results in an error.) Under this rule, [B/D] would deduce that $X_t = V_t + V_{t-1} + \ldots + V.1$, as X_0 is taken to be zero. Otherwise, termination criteria can be set explicitly by defining an element. This works as follows.

An assignment is defined as a linear combination of elements, components, and other assignments. Therefore, and ignoring self-referential assignments which cause an error, it is possible to trace through the definition part until it is solely in terms of elements and components.

When an assignment is referenced, [B/D] tries to determine the linear combination of elements and components intended in the definition part of the assignment. If the definition part of the assignment contains other assignments, these too are tracked through to determine the intended linear combination of elements and components. Eventually all the assignments in the definition part will either vanish because they are replaced by their own definitions, or as the recursion nears zero, they become zero by default, or because of terminating criteria established beforehand. Consider, for example, the following fragment of code (#phi is an example of a function, described more fully in chapter 7).

BD>**element: A, X.1, Z.2** ↩
BD>**assign: X.t=X.(t-1)+V.t** ↩
BD>**assign: Z.t=(#phi) Z.(t-1)+E.t** ↩
BD>**assign: Y.t=X.t+Z.t+ A** ↩

When the assignment $Y.t$ is referenced, [B/D] will take into account the recursion involving $X.t$ and $Z.t$. Moreover, as the elements $X.1$ and $Z.2$ exist, these will take priority over the implied assignments of the same name, and the recursion will terminate at these points. For example, suppose that we now wished to construct $Y.3$. The assignments and termination criteria imply

$$Y.3 = A + X.1 + V.2 + V.3 + \phi Z.2 + E.3$$

6.4.6 Deleting assignments

▷ ▷ **Syntax**

BD>**xassign:** A_1 [, A_2 ...] ↩

where $A_1, A_2, \ldots$ *are the names of* assignments. *Only the part of the name preceding indices (if any) should be given.*

◁ ◁

The XASSIGN: command is used to remove assignments.

6.5 Defining and using collections

6.5.1 Overview

The *base* is the principal organising tool in [B/D]. Elements, the belief carrying quantities, are organised into *bases*, which thus become the main focus of attention. We consider elements to be bases in there own right, and from the point of view of adjustments we adjust bases rather than elements, and our summaries refer globally to these collections of elements rather than to the elements themselves (although, of course these details are available for the elements individually).

The BASE: command is used to declare bases, and the XBASE: command is used to remove them. The LOOK: command can be used to inspect some or all of them. See §6.2 for details of specifying collections for the definition part.

6.5.2 Declaring bases

▷ ▷ **Syntax**

1. BD>**base: B0=B1,B2,** ... ↩

2. BD>**base: B0=B1,B2^,** ... ↩

where $B0$ *is the name of the base being defined, and* $B1, B2\ldots$ *are the names of* elements *and* bases.

◁ ◁

The BASE: command is used to define a *base*, being a collection of elements (and possibly data carriers). The definition part consists of the names of the elements and data carriers to form the base. If the names of any pre-existing bases are included in the definition part, their constituents are included in the new base $B0$. If the base being defined in the naming part also appears in the definition part, all the elements included in the former definition are also included in the new definition. Unrecognised quantities are allowed in the RHS of the definition. This allows semi-recursive base creation, as in

BD>**base: B1=B1,B2** ↩

where B1 does not exist a priori.

The second form of the syntax is used to *exclude* consituents of the base constructed so far. See §6.2.3.

6.5.3 Other remarks

1. Bases are generally deleted by using the XBASE: command discussed below. However, whenever a base becomes redundant because all its constituents have been deleted using the XELEMENT: command, the base too is removed.

2. As far as possible we take no account of the ordering in a collection: an element either belongs to a base, or not, and we prefer to regard its position in the base as not of interest. (The important exception is for the input of belief matrices using the VAR: command.) However, it is possible to refer to the i^{th} member of a base according to the ordering convention using the notation [<B:I>], where B is the name of the base, and I is an integer. The effect of this notation is as follows. Every line input to [B/D] is parsed before any action is taken. A quantity of the form [<B:I>] is replaced by the name of the element or data carrier having position I in the base B. You should beware using this facility as it depends crucially upon the ordering convention (this is discussed in the glossary), and this will - of course - change as elements and data carriers are introduced and deleted.

3. Two operators are available to count the number of bases and elements in bases. The **count** operator counts the number of elements, but not bases or data carriers, in a base. The **countb** operator counts the number of either elements or bases, but not data carriers, in a base. The **countd** operator counts only data-carriers, and excludes elements possessing data. The **countany** operator counts all constituents of a *base*, i.e. data-carriers, elements, and other bases.

4. It is possible to define bases of bases. In particular, a base of bases can be used during adjustments. See §9.1.2 for further details.

6.5.4 Deleting bases

▷ ▷ **Syntax**

BD>**xbase:** B_1 [, B_2 …] ↩

where $B_1, B_2, \ldots$ *are the names of* bases.

◁ ◁

The XBASE: command is used to delete the bases named as arguments to the command. When a base is deleted, and if this base is the name of a node defined via the GRID: command for influence diagram use, then the node is deleted, and any arcs connecting this node to any other are deleted. See §6.2 for details of specifying collections for the definition part.

6.6 Declaring and using elements

6.6.1 Overview

The fundamental belief-carrying unit is the *element*, which corresponds to random variable in traditional statistical packages. These *elements* may be organised into collections which we term *bases*, and we usually think of elements as bases in their own right. Elements are the only kind of quantity for which we specify variances, covariances and expectations explicitly, and consequently our principal interest lies in the relationships specified over and between collections of elements, and any observations pertaining. Much of the computation in [B/D], and in particular the concept of adjustment, is aimed at elements, Thus, for example, only elements and collections of them can be adjusted.

Elements are introduced into [B/D] by the ELEMENT: command directly, or are constructed from linear combinations of other quantities via the BUILD: and COBUILD: commands. Expectations for elements can be introduced explicitly at the time of the definition within the ELEMENT: command, or may be altered at any stage via the E: command. Beliefs are typically specified over elements via the VAR: command directly, or are calculated automatically from their defining linear combination if they are constructed from other quantities. Data can be attached directly to elements using the DATA: command, or is automatically calculated if the element is constructed from other elements. Elements can be removed from [B/D] by using the XELEMENT: command.

Finally, elements are *always* associated with expectations, variances, and covariances with other elements, even if these values have not been specified – they are taken to be zero by default. Consequently, elements take up quite a bit of machine memory, although they are usually handled very quickly.

As an example, typical elements might be the temperature, T, and blood pressure, P, of some patient, organised into some base X. Associated with these quantities are expectations and covariances as follows:

$$\mathrm{E}(X) = \begin{bmatrix} \mathrm{E}(T) \\ \mathrm{E}(P) \end{bmatrix}, \qquad \mathrm{Var}(X)_{(1)} = \begin{bmatrix} \mathrm{Var}(T) & \mathrm{Cov}(T,P) \\ \mathrm{Cov}(P,T) & \mathrm{Var}(P) \end{bmatrix}.$$

We show the subscript (1) to indicate that we might like to associate with a collection of elements *more than one* variance–covariance specification. We frequently make use of this facility when we deal with exchangeable sequences. We might also have actual observations, perhaps repeated, of T and P.

6.6.2 Alternative expectation stores

There are circumstances in which it is convenient to be able to specify alternative vectors of expectations (and alternative covariance matrices) for elements. Thus, for both expectations and variance-covariance specifications, we allow a number of alternative stores - although the two kinds of store mostly work completely independently. For simplicity and backwards compatibility we also introduce the notion of default expectation store. The number of expectation stores is generally fixed: it is the same as the number of belief stores, and this can be discovered by executing the LOOK: command with argument program; for some current versions of [B/D] it is 6. The default expectation store number is number one. Most of the commands which reference element expectations work with expectations from the default store number. Such commands include the ADJUST:, SCAN:, ITADJUST:, and COMPARE: commands. Other commands, essentially those involved in defining or modifying expectations or covariances - such as the BUILD:, COBUILD:,CENTRE: and PRODUCT: commands - deal specifically with individual expectation stores.

A brief summary of the commands and controls affecting expectation stores is as follows. Alternative expectation stores may be locked or unlocked in manner similar to belief stores by using the ELOCK: command. The default expectation store number can be changed using the e argument to the CONTROL: command. Expectations from the various stores can be examined using the e argument to the LOOK: command, whilst this and the the nc argument reveal which expectation store is currently the default. Expectations from alternative stores can be accessed via the ex operator. Adjusted expectations resulting from ADJUST: or SCAN: commands can optionally be retained in an expectation store using the ae and scae controls. Components may have different functional expectations specified in different expectation stores using the FE: command. The EXP: command is used to define expectations for alternative expectation stores. Expectations from any given expectation store can be exported using the EXPORT: command.

6.6.3 Using elements to represent exchangeable sequences

Consider a second–order exchangeable sequence $X_1, X_2, \ldots$. This has the property (see [37]) that each X_i may be decomposed as $X_i = \mathcal{M}(X) + \mathcal{R}_i(X)$, where

$$\mathcal{M}(X), \mathcal{R}_1(X), \mathcal{R}_2(X), \ldots$$

are uncorrelated; and where the $\{\mathcal{R}_i(X)\}$ have expectation zero and common variance σ_R^2. For such sequences it is typically possible to learn only about the common mean quantity M. Therefore, the sequence can be summarised by knowledge of $\mathrm{E}(M)$, $\mathrm{Var}(M)$, σ_R^2, and the notion of repetition.

In [B/D] we choose to represent an exchangeable sequence by a solitary *element* with expectation $\mathrm{E}(M)$ being the expectation common to each X_i. For the variance quantities $\mathrm{Var}(M)$ and σ_R^2 (the sum $\mathrm{Var}(M) + \sigma_R^2$ is the common variance $\mathrm{Var}(X_i)$) we associate with the element two different variance storage areas. For example, to summarise this exchangeable sequence we could define an element called X with associated expectation $\mathrm{E}(X) = \mathrm{E}(M)$, and specify two kinds of variances: $\mathrm{Var}(X)_{(1)} = \mathrm{Var}(M)$ and $\mathrm{Var}(X)_{(2)} = \mathrm{Var}(M) + \sigma_R^2$. (it is usually more covenient to store the overall variance, as the residual variance can always be found by subtraction if necessary).

6.6.4 Declaring elements

▷ ▷ **Syntax**

1. `BD>`**element:** [<**B**>] $\mathbf{N}_1$[=$\mathbf{E}_1$] [,$\mathbf{N}_2$[=$\mathbf{E}_2$]], ... ↩

2. `BD>`**element:** [<**B**>] ↩

3. `BD>`**element:** [<**B**>] **@L** ↩

4. `BD>`**element:** [<**B**>] **@L(C)** ↩

5. `BD>`**element:** [<**B**>] **(C)** ↩

where B is the name of a base *which may or may not already exist, $N_1, N_2, \ldots$ are the names of the* elements *being defined, $E_1, E_2, \ldots$ are valid* equations*, L is the name of a* label *and $0 \leq C \leq 6$ is a valid* input channel number.

◁ ◁

The first form of the syntax is the most usual, and consists of the command followed by a number of *element* names $N_1, N_2, \ldots$, optionally followed by an expectation E_i for the element. Where no expectation is explicitly given, the expectation is taken to be zero by default. The expectation is for the default expectation store: the default can be changed by using the e argument to the CONTROL: command. The value given for the expectation may be changed at a later date by using the E: or EXP: commands. Expectations for this element in other expectation stores are set to zero. Optionally, the elements declared may be gathered into the base named B, which may or may not already exist. If the base B already exists it is overwritten.

If an element name already exists, it is overwritten, and its expectation in the default expectation store is as given in the ELEMENT: command, or defaults to zero, with expectations in other stores being set to zero. However, it adopts the variance and covariance specifications carried by the former element. To clear the variance and covariance specifications, issue the XELEMENT: command first.

The remaining forms of the syntax are used to input a list of elements either interactively or from a specified address on a macro file. As for the first form of the syntax, optionally the elements may be gathered into the base B, and expectations may or may not accompany the element names, with zero expectations being the default. The second form of the syntax is used for interactively inputting a list of elements. The remaining forms specify an address where a list of elements may be found. If the third form of the command is issued, with no channel number being specified, the channel number is deduced - [B/D] does not allow duplicate labels. In the fifth form of the syntax, the beginning of the file associated with the input channel is indicated. In common with other macro facilities, the channel number can be zero (meaning the keyboard): a label should not be supplied in this case. The channel and label specified or implied must exist at the time the command is issued. An example of the format necessary for the list of elements to be input is as follows:

@example These are elements with some non-zero expectations ↩
x1, x2=19.3 ↩
y1 ↩
z1=19.6, z2=20.1, alpha ↩

↩

The element list is terminated by the # symbol. This is true also for interactive input of a list of elements. Suppose that the elements listed in the example just given are listed on some macro file already announced to the program. They could now be input to [B/D] and gathered into the base named *xyz* by issuing the following command:

`BD>`**element: <xyz> @example** ↩

6.6.5 Changing expectations

▹ ▹ **Syntax**

`BD>`**e: $\mathbf{N}_1$=$\mathbf{E}_1$, $\mathbf{N}_2$=$\mathbf{E}_2$, ...** ↩

where $N_1, N_2, \ldots$ are the names of elements*, and $E_1, E_2, \ldots$ are valid* equations*.*

◃ ◃

The E: command is used to *change* current expectations for the default expectation store. It cannot be used to introduce or redefine an element name, and does not affect variance or covariance specifications. It is an error to attempt to change the expectation for an element which does not exist. The result of the command is to change the expectation for each N_i from what it was to the value given by the equation E_i. To change expectations for otherv than the default expectation store, use the EXP: command. To change the default expectation store, use the e argument to the CONTROL: command.

6.7 Declaring expectations for an explicit expectation store

6.7.1 Overview

There are two basic ways of declaring expectations for a given expectation store. We may (1) declare them as the value of some given *equation*; (2) directly input numerical specifications. We treat each of these methods separately below, although the same EXP: command is used for each. Expectations attach to predefined elements only, and are at least notionally available as soon as an element becomes defined. By default, all expectations are zero unless specified otherwise.

6.7.2 Declaring single expectations

▹ ▹ **Syntax**

`BD>`**exp: e(S,N)=E** ↩

where N is the name of an elements *which must already exist, E is a valid* equation*, and S is an* expectation store number*.*

◃ ◃

The syntax for the EXP: command given here is used for the declaration of single expectations. The result of the command is to specify the value of the equation E for the expectation of N in expectation store S. Two examples are

`BD>`**exp: e(2,temperature)=2.5** ↩
`BD>`**exp: e(1,X)=ln(%c)+ln(%d)** ↩

6.7.3 Direct numerical specification of alternative expectations

▹ ▹ **Syntax**

1. `BD>`**exp: e(S,B)** ↩

2. BD>**exp: e(S,B) @L** ↩

3. BD>**exp: e(S,B) @L(C)** ↩

4. BD>**exp: e(S,B) (C)** ↩

where B *is the name of a* base *which must already exist,* S *is a* belief store number*,* L *is the name of a* label *and* $0 \leq C \leq 6$ *is a valid* input channel number*.*

◁ ◁

The particular forms for the syntax of the EXP: command given here all relate to the direct input of a vector of numerical expectation specifications. The number of numerical inputs supplied depends only on the dimension of the base B. The form of the syntax used establishes the place from which the beliefs will be sought: interactively or from a given macro file address.

Firstly we consider the placing of the numerical inputs. Form 1 of the syntax is used when the inputs follow directly afterwards. Therefore, this form of the syntax can be used to obtain expectations interactively from the keyboard. The remaining forms specify an address where the numeric inputs should be found. If form 2 of the syntax is used, with no channel number being specified, the channel number is deduced - [B/D] does not allow duplicate labels. In form 4 of the syntax, the beginning of the file associated with the input channel is indicated. In common with other macro facilities, the channel number can be zero (meaning the keyboard): a label should not be supplied in this case. The channel and label specified or implied must exist at the time the command is issued.

If the base B contains n elements, then the program expects n numbers to be input as their corresponding expectations. These must be supplied according to alphabetical ordering of the elements contained in B, but you are free to split the input over several lines, as many numbers per line as you wish (possibly interspersed with blank lines) subject to the usual limitation of no more than 253 characters per physical line. A number must not be split over two or more lines. Any input on the same physical line as the last number required will be ignored. The numbers are to be in standard numerical format; an error message is printed if the number is invalid, and the command is aborted. It is advisable not to input large vectors interactively. As an example, suppose that $B = x_1, x_2, x_3, x_4$, with intended expectations 0.1, 0.2, 0.3, 0.4 respectively to be put into expectation store 2. This could be achieved by the following command:

BD>**exp e(2,B)** ↩
0.1 ↩
0.2 0.3 ↩
0.4 ↩

As a second example, suppose that we wish to read the vector of expectations from the file "test.data". This might be achieved, for example, by issuing the commands

BD>**channel i3=test.data** ↩
BD>**exp e(2,B)(3)** ↩

6.7.4 Deleting elements

▷ ▷ **Syntax**

BD>**xelement:** B_1 [, B_2 ...] ↩

where $B_1, B_2, \ldots$ *are the names of* elements *or* bases*.*

◁ ◁

This command deletes elements. The arguments specify a collection of elements which are to be deleted. Any bases whose names appear in the argument list are not deleted unless all their elements are deleted. Wildcards may be used with this command. See §6.2 for details of specifying collections for the definition part.

If XELEMENT: is requested to remove a specific element which it fails to recognise, a warning is reported.

The contents of bases are always checked after elements have been deleted, and if any bases have become redundant as a result, they are also deleted.

When an element is deleted, and if this element is the name of a node defined via the GRID: command for influence diagram use, then the node is deleted, and also deleted are any arcs connecting this node to any other node.

6.8 Declaring variances and covariances numerically

6.8.1 Overview

There are many ways of declaring variances and covariances over and between elements. We may (1) declare single variances or covariances as the value of some given *equation*; (2) construct variance-covariance matrices as linear combinations of other variance-covariance matrices; (3) directly input numerical specifications. We treat each of these methods separately below, although the VAR: command is used for each. Variances and covariances attach to elements only, and are at least notionally available as soon as an element becomes defined. By default, all variances and covariances are zero unless specified otherwise.

Individual belief specifications are available for use in equations in either their original covariance form or in correlation form. Use the **var** and **corr** operators respectively. All belief specifications can be examined by using the LOOK: command with either the v (for covariances) or r (correlation form) argument.

Belief specifications should, of course, be coherent. However, no check is made at the time of specification as to whether the inputs are coherent. Generally this is left until the beliefs are actually used as inputs to an adjustment. It is possible to use the COHERENCE: command to verify belief coherence directly, and less directly the EIGEN: command may also be used to this purpose.

6.8.2 Declaring single covariances

▹ ▹ **Syntax**

1. BD>**var: v(S,N$_1$,N$_2$)= (E)** ↩

2. BD>**var: v(S,N$_1$)=(E)** ↩

where $N_1, N_2, \ldots$ are the names of elements *which must already exist, E is a valid* equation*, and S is a* belief store number.

◃ ◃

The syntax for the VAR: command given here is used for the declaration of single variances and covariances. The first form of the syntax is used to specify the covariance between the elements N_1 and N_2 in belief store number S to be the value of the equation E. The round brackets surrounding the equation are essential unless E is a real number. The two names N_1 and N_2 are allowed to be the same, implying of course the declaration of a variance. The second fom of the syntax is a special short form of the first syntax when $N_2 = N_1$ and we wish to specify a variance. Examples are:

BD>**var: v(2,temperature,bloodpressure)=2.5** ↩
BD>**var: v(1,X)=(ln(%c)+ln(%d))** ↩

6.8.3 Constructing variance matrices from other variance matrices

▹ ▹ **Syntax**

BD>**var: v(S$_1$,B$_1$,B$_2$)=[(E$_1$)][v(S$_2$,B$_3$,B$_4$)] ± . . . ± [(E$_2$)][r(S$_3$,B$_5$,B$_6$)] ± (E*) ± . . .** ↩

where B_1, B_2 *are the names of* bases *which must already exist,* $E^*, E_1, E_2, \ldots$ *are valid* equations*, and* $S_1, S_2, \ldots$ *are* belief store numbers.

◁ ◁

This usage of the VAR: command is used to construct variance matrices from linear combinations of other variance and/or correlation matrices. The naming part indicates that the beliefs being specified are the covariances for belief store number S_1 between the collection of elements B_1 and the collection of elements B_2. (If $B_2 = B_1$ then $, B_2$ may be omitted.) The optional coefficients in the linear combination are the equations $E_1, E_2, \ldots$ which must be enclosed within round brackets.

The matrices in the linear combination are either variance matrices or correlation matrices. For example, the syntax v(S_2,B_3,B_4) refers to the variance-covariance matrix specified between the collection of elements B_3 and the collection of elements B_4 for belief store number S_2. (If $B_3 = B_4$ then $, B_4$ may be omitted.) r(S_3,B_5,B_6) refers to the correlation matrix specified between the collection of elements B_5 and the collection of elements B_6 for belief store number S_3. (If $B_5 = B_6$ then $, B_6$ may be omitted.) Where a correlation matrix is indicated, the corresponding covariance matrix is accessed and the correlations calculated temporarily.

The matrices involved in the command must be conformable. You should be careful when you use commands of this kind when the collections in the definition part are neither the same as, nor disjoint to, the collections in the naming part. This is because [B/D] assumes all variance-covariance matrices to be symmetric and thus stores only part of the matrix. Consequently, asymmetries in what you attempt may result in unintended misspecifications. See §6.8.5 for an illustration of this kind of problem. These problems do not arise when (1) (B_1, B_2) is disjoint to (B_3, B_4) and (2) $(B_1, B_2) = (B_3, B_4)$, and similarly for any other pair of collections appearing in the definition part. It is perfectly possible for $B_1 = B_3 = B_5$ and $B_2 = B_4 = B_6$ as the definition part is evaluated entirely before being stored.

If an equation such as E^* is found without being the coefficient of a succeeding matrix, a matrix of conformable dimensions of scalars all equal to the value of the equation E^* is added or subtracted as necessary.

6.8.4 Direct numerical specification of variance matrices

▷ ▷ **Syntax**

1a `BD>`**var: v(S,B$_1$,B$_2$)** ↩

1b `BD>`**var: v(S,B$_1$)** ↩

2a `BD>`**var: v(S,B$_1$,B$_2$) @L** ↩

2b `BD>`**var: v(S,B$_1$) @L** ↩

3a `BD>`**var: v(S,B$_1$,B$_2$) @L(C)** ↩

3b `BD>`**var: v(S,B$_1$) @L(C)** ↩

4a `BD>`**var: v(S,B$_1$,B$_2$) (C)** ↩

4b `BD>`**var: v(S,B$_1$) (C)** ↩

where B_1, B_2 *are the names of* bases *which may or may not already exist,* S *is a* belief store number, L *is the name of a* label *and* $0 \le C \le 6$ *is a valid* input channel number.

◁ ◁

The particular forms for the syntax of the VAR: command given here all relate to the direct input of a body of numerical variance and covariance specifications. The number of numerical inputs supplied depends upon both the dimensions of the vectors involved and the form of syntax used. The form of the syntax used also establishes the place from which the beliefs will be sought: interactively or from a given macro file address.

Firstly we consider the placing of the numerical inputs. Forms 1a and 1b of the syntax are used when the inputs follow directly afterwrads. Thus, this form of the syntax may be used to obtain beliefs interactively from the keyboard. The remaining forms specify an address where the numeric inputs should be found. If forms 2a and 2b of the command are used, with no channel number being specified, the channel number is deduced - [B/D] does not allow duplicate labels. In forms 4a and 4b of the syntax, the beginning of the file associated with the input channel is indicated. In common with other macro facilities, the channel number can be zero (meaning the keyboard): a label should not be supplied in this case. The channel and label specified or implied must exist at the time the command is issued.

Secondly we consider the formatting of the inputs. The 1a, 2a, 3a, and 4a forms are used to input a rectangular matrix of belief specifications for the covariances between the collections B_1 and B_2. Suppose, for example, that B_1 consists of the elements $[X_1 \dots X_n]$ and that B_2 consists of the elements $[Y_1 \dots Y_m]$. (It is possible for B_2 to be the same as B_1.) The belief inputs consist of the following specifications:

$$\begin{bmatrix} \mathrm{Cov}(X_1,Y_1) & \cdots & \mathrm{Cov}(X_1,Y_m) \\ \mathrm{Cov}(X_2,Y_1) & \cdots & \mathrm{Cov}(X_2,Y_m) \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n,Y_1) & \cdots & \mathrm{Cov}(X_n,Y_m) \end{bmatrix}.$$

For smaller specifications at least, we recommend that you input the numbers required as an array of n rows each consisting of m columns, with the numbers separated by one or more spaces. However, the real numbers input need not necessarily be in tabular format They must be supplied in the order indicated by reading the matrix above from left to right, top to bottom, but you are free to split the input over several lines, as many numbers per line as you wish (possibly interspersed with blank lines) subject to the usual limitation of no more than 253 characters per physical line. A number must not be split over two or more lines. Any input on the same physical line as the last number required will be ignored. The numbers are to be in standard numerical format; an error message is printed if the number is invalid, and the command is aborted. It is advisable not to input large matrices interactively.

As an example, suppose that $B_1 = x_1, x_2, x_3$ and that $B_2 = y_1, y_2$ and that the following covariance matrix is given on a macro file:

@xybeliefs This is the 3x2 covariance matrix between the x's and the y's ↩
1.0 -3.2 ↩
0.5 2.6 ↩
1.0 -2.2 ↩

We could now input this covariance matrix into belief store number 2 by issuing the command

BD>**var v(2,B1,B2) @xybeliefs ↩**

We now consider the remaining forms 1b, 2b, 3b, and 4b, which are used to input the lower triangle of the square symmetric variance-covariance matrix specified over the collection B_1. Suppose, for example, that B_1 consists of the elements $[X_1 \dots X_n]$. The belief inputs consist of the following specifications:

$$\begin{bmatrix} \mathrm{Var}(X_1) & & & \\ \mathrm{Cov}(X_2,X_1) & \mathrm{Var}(X_2) & & \\ \vdots & \vdots & & \\ \mathrm{Cov}(X_n,X_1) & \mathrm{Cov}(X_n,X_2) & \cdots & \mathrm{Var}(X_n) \end{bmatrix}.$$

That is there are n rows. The first row has one number, the second row has two numbers, and so forth. For smaller specifications at least, we recommend that you input the $n(n+1)/2$ numbers required as a triangle with n rows, and with the numbers separated by one or more spaces. However, the real numbers input need not necessarily be in this format. They must be supplied in the order indicated by reading the matrix triangle above from left to right, top to bottom, but you are free to split the input over several lines, as many numbers per line as you wish (possibly interspersed with blank lines) subject to the usual limitation of no more than 253 characters per physical line. A number must not be split over two or more lines. Any input on the same physical line as the last number required will be ignored. The numbers are to be in standard numerical format; an error message is printed if the number is invalid, and the command is aborted. It is advisable not to input large matrices interactively.

As an example, suppose that $B_1 = x_1, x_2, x_3$ and that we interactively input its variance matrix into belief store number 1. The following sequence of commands suffice:

BD>**var: v(1,B1)** ↩
BD***5.3** ↩
BD***0.9 4.6** ↩
BD***1.1 1.3 7.2** ↩

Notice that the [B/D] prompt changes from the '>' character to the '-' character when the program is expecting interactive inputs.

6.8.5 Problems of asymmetry

The beliefs that you specify over pairs and so forth must be symmetric: that is, for all X, Y we must specify $\text{Cov}(X, Y) = \text{Cov}(Y, X)$. An illustration of the kind of problem that might arise is given below.

BD>**element: a,b,c,d,e,f** ↩
BD>**base: x=a,c,d,e** ↩
BD>**base: y=b,c,d,f** ↩
BD>**var: v(2,x,y)** ↩
BD***-2 -1 -1 -6** ↩
BD***-1 0 6 -5** ↩
BD***-1 5 0 -5** ↩
BD***-6 -5 -5 -6** ↩

This sequence of commands specifies a 4×4 covariance matrix between the collections x and y. However, x and y are not disjoint - they both contain the elements c and d - and the actual specification mistakenly attempts to define $\text{Cov}(c, d) = 6$ and $\text{Cov}(d, c) = 5$. An error is flagged whenever this kind of mistake occurs. This problem can arise also when using the VARDATA: and MATMULT: commands.

6.9 Functional belief specifications

6.9.1 An introduction to components

Components are notional elements. They differ from true elements in that they have no formal existence except for functional relationships specified over themselves and between themselves and true elements. Provided that the number of such functional relationships is quite limited, this is a memory–efficient way of handling belief carriers, particularly as large numbers of quantities differing only in index, say, can be represented as only one component. However, they are often handled quite slowly, and they cannot be adjusted. Their principal role is as intermediaries from which we construct elements which can be adjusted.

Components only exist via functional relationships expressed. Functional belief specifications are attached to components as follows. The FE: command is used to define their expectations, and may involve a function of some index. Beliefs are expressed over components as functions, using the FVAR: command. This command is also use for the specification of beliefs between different components and between components and elements. Again, these might involve functions of indices. A components might be associated with data in that there may be a *data carrier* of the same name.

Functional belief specifications can be removed using the XFVAR: command; and functional expectations can be removed by using the XFE: command.

As as a simple example, consider a sequence of quantities $\eta_1, \eta_2, \ldots$, which we intend to represent some error quantities. Suppose that these quantities have expectation zero, are uncorrelated with all other quantities, and have the following covariance structure:

$$\text{Cov}(\eta_i, \eta_j) = \begin{cases} 1 & i = j \\ \frac{1}{i+j} & \text{otherwise.} \end{cases} . \tag{6.1}$$

Then all these quantities can be represented succinctly in [B/D] by one component, E say, with an index, for which the covariance structure can be expressed as a simple function.

Individual functionally specified beliefs can be accessed in the same way as directly specified beliefs, that is via the **ex**, **var**, and **corr** operators.

6.9.2 Declaring functional variances and covariances

▷ ▷ **Syntax**

BD>**fvar: v(S, X [IL$_1$], Y [IL$_2$]) = E** ↩

where S is a belief store number*, X and Y are the names of* components *or* elements*, IL_1, IL_2 are* index lists*, and E is an equation, possibly involving indices from either index list.*

◁ ◁

The FVAR: command is used to specify variances and covariances functionally for components and between components and elements. At least one of X and Y should be a component, but one may be an element. The functional belief being defined is expressly limited to belief store number S, and will overwrite any functional belief expressed for the same pair of quantities for the same belief store. However, further functional beliefs could be expressed about the same pair of quantities for different belief store numbers.

The *index list* comprises single alphabetic characters, preceded by the '.' symbol. An alphabetic character must not appear more than once in the entire naming part, and the number of indices per component is currently limited to 8.

The equation E will usually relate to the as yet unknown indices of the naming part. As an example, consider the quantities $\{\eta_i\}$ discussed in (6.1) above, and suppose that we wish to specify the variance-covariance structure for belief store number 1 for these quantities. We could use the following command to achieve this:

BD>**fvar: v(1,eta.i,eta.j)=(.i=.j)+(.i<>.j)/(.i+.j)** ↩

6.9.3 Deleting functionally specified variances and covariances

▷ ▷ **Syntax**

BD>**xfvar:** X_1, X_2 [, X_3, X_4 ...] ↩

where $X_1, X_2, \ldots$ are the names of quantities to which functional variances or covariances have been attached. Only the part of the name preceding indices (if any) should be given.

◁ ◁

The XFVAR: command is used to remove functionally specified variances and covariances. Each *pair* of arguments defines a functional specification which is to be deleted. (This corresponds to the insistence on defining functional beliefs for a named pair only.)

6.9.4 Declaring functional expectations

▷ ▷ **Syntax**

BD>**fe: e(S,X [IL]) = E** ↩

where X is the name of a component*, IL is an* index list*, S is an expectation store number, and E is an equation, possibly involving indices from the index list.*

◁ ◁

The FE: command can be used to attach functional expectations to *components*, and these are accessed when the corresponding components are the subject of BUILD:, COBUILD: and ITADJUST: commands to specify expectations for the component quantities. Otherwise, components are assumed to have zero expectation. These functionally defined expectations are deleted by the XFE: command. An expectation store number must be given.

As an example, suppose that we choose to ensure that the η_i of (6.1) have expectations equal to i^2 in expectation store 1, and expectation $i^2 + 1$ in expectation store 2. This is achieved by the following commands:

BD>**fe: e(1,eta.i)=sqr(.i)** ↩
BD>**fe: e(2,eta.i)=sqr(.i)+1** ↩

A quantity may be associated with several alternative functional expectations in different expectation stores. In this case, the index lists should be compatible, i.e. of the same length. The following example shows proper use of this facility:

BD>**fe: e(1,b.i.j)=sqr(.i)+.j** ↩
BD>**fe: e(2,b.i.j)=sqr(.i)-.j** ↩

6.9.5 Deleting functionally specified expectations

▹ ▹ **Syntax**

BD>**xfe:** X_1 [, X_2 ...] ↩

where $X_1, X_2, \ldots$ *are the names of quantities to which functional expectations have been attached. Only the part of the name preceding indices (if any) should be given.*

◃ ◃

The XFE: command is used to remove functionally specified expectations given using the FE: command. If the quantity is associated with alternative expectations in different expectation stores, all are deleted.

6.10 Constructing beliefs from products of quantities

▹ ▹ **Syntax**

BD>**product: A, B** ↩

where A and B are the names of bases

◃ ◃

The PRODUCT: command is used for the construction of beliefs over products of uncertain quantities under certain conditions. These conditions are as follows. Suppose that the base A consists of the elements $A_1, A_2, \ldots, A_n$ and that the base B consists of the elements $B_1, B_2, \ldots, B_m$. Suppose also that we gather all the elements not in either A or B into the base C, and suppose that we label these elements $C = C_1, C_2, \ldots, C_r$.

1. The bases A and B must be disjoint. That is, the elements of the base A must not also appear in the collection B.

2. The collection A must be uncorrelated with the collection B for every *unlocked* belief store. That is, we must have $\mathrm{Cov}(A_i, B_j) = 0$ for all i, j. An error is reported otherwise.

3. For pairs of quantities $A_i \in A, B_j \in B$, and for each $C_k \in C$ we must have either

$$\mathrm{Cov}(A_i, C_k) = 0 \quad \text{or} \quad \mathrm{Cov}(B_j, C_k) = 0$$

or both, for every unlocked belief store. An error is reported otherwise.

The product formed by the pair A_i, B_j will be constructed as the element named 'Ai~Bj', for all $A_i \in A$ and all $B_j \in B$. Any preexisting elements with these names will be deleted, together with any data attached to them. This is unlike the BUILD: command in that data attached to a pre-existing element is not here adopted by the replacing element.

The PRODUCT: command is asymmetric to the extent that the command

BD>**product: B, A** ↩

will construct different names (of the form 'Bj˜Ai') for the same collection of products. Hence you should take care with the order given in the command.

Beliefs over the products are constructed only for *unlocked belief stores*, and the uncorrelatedness conditions relate only to these unlocked stores. Expectations over products are constructed only for unlocked expectation stores. The LOCK: and ELOCK: commands may be used to lock and unlock stores.

If the builddata control is switched on, the PRODUCT: command also constructs data from the product wherever possible. Similarly to the construction of data for the BUILD: command, the maximum number of cases for which data is available for both A_i and B_j is used to construct the data for the new element $A_i\tilde{\ }B_j$

The following formulae are used to construct beliefs over the product, and between the product and other elements.

$$\begin{aligned}
\mathrm{Cov}(A_i\tilde{\ }B_j, A_k\tilde{\ }B_t) &= (\mathrm{Cov}(A_i, A_k) + \mathrm{E}(A_i)\mathrm{E}(A_k))\mathrm{Cov}(B_j, B_t) + \mathrm{Cov}(A_i, A_k)\mathrm{E}(B_j)\mathrm{E}(B_t) \\
\mathrm{Cov}(A_i\tilde{\ }B_j, A_k) &= \mathrm{Cov}(A_i, A_k)\mathrm{E}(B_j) \\
\mathrm{Cov}(A_i\tilde{\ }B_j, B_t) &= \mathrm{Cov}(B_j, B_t)\mathrm{E}(A_i) \\
\mathrm{Cov}(A_i\tilde{\ }B_j, C_k) &= \begin{cases} \mathrm{Cov}(A_i, C_k)\mathrm{E}(B_j) & \text{if} \quad \mathrm{Cov}(C_k, B_j) = 0 \\ \mathrm{Cov}(B_j, C_k)\mathrm{E}(A_i) & \text{if} \quad \mathrm{Cov}(C_k, A_i) = 0 \\ 0 & \text{if} \quad \mathrm{Cov}(C_k, B_j) = 0 \quad \text{and} \quad \mathrm{Cov}(C_k, A_i) = 0 \end{cases}
\end{aligned}$$

6.11 Related commands

6.11.1 Forming covariances from beliefs expressed as cross-products

▹ ▹ **Syntax**

BD>**centre: B** ↩

where B is the name of a base *or* element.

◃ ◃

The CENTRE: command is used to transform cross product belief specifications into covariances. That is, for each pair of elements N_1 and N_2 in B, and for each of the *unlocked* belief stores, S, the current value stored for the specification over the pair (N_1, N_2) is replaced by this value minus the product of the respective expectations for N_1 and N_2. This command should be used when a set of belief specifications is given in the form $\mathrm{E}(XY^T)$ rather than the form $\mathrm{Cov}(X, Y) = \mathrm{E}(XY^T) - \mathrm{E}(X)\mathrm{E}(Y)^T$, where X and Y are vectors of unknowns. See §6.2 for details of specifying collections for the definition part.

6.11.2 Locking and unlocking belief storage areas

▹ ▹ **Syntax**

BD>**lock:** [-](**S**$_1$) [, [-](**S**$_2$)] ... ↩

where $S_1, S_2, \ldots$ are belief storage numbers.

◃ ◃

The LOCK: command is used to lock and unlock specified belief storage areas. If the argument S_1 is preceded by the symbol '-', the belief store number S_1 is 'unlocked', otherwise it becomes 'locked'. When a belief store is locked, beliefs usually constructed via BUILD: or COBUILD: commands are not constructed, and so do not overwrite the values already present. It remains possible to specify directly beliefs for locked storage areas. By default, all belief storage areas are unlocked. The **stores** operand can be used to obtain the total number of belief stores.

The LOCK: command is often used when we want to build two alternative variance-covariance specifications for a collection of quantities in stages, and also when we have output adjusted covariances to a belief storage area which we want to be retained.

Beware that some controls force the unlocking of specified belief stores. In particular, when the ac and scac controls are used to direct adjusted covariance output from an adjustment or a scan to a belief store, the given belief store is unlocked.

6.11.3 Locking and unlocking expectation storage areas

▷ ▷ **Syntax**

BD>**elock:** [-]$(\mathbf{S}_1)$ [, [-]$(\mathbf{S}_2)$] ... ↩

where $S_1, S_2, \ldots$ *are expectation storage numbers.*

◁ ◁

The ELOCK: command is used to lock and unlock specified expectation storage areas. If the argument S_1 is preceded by the symbol '-', the expectation store number S_1 is 'unlocked', otherwise it becomes 'locked'. When an expectation store is locked, expectations usually constructed via BUILD: or COBUILD: commands are not constructed, and so do not overwrite the values already present. It remains possible to specify directly expectations for locked storage areas. By default, all expectation storage areas are unlocked. The **stores** operand can be used to obtain the total number of expectation stores.

The ELOCK: command is often used when we want to build two alternative expectation specifications for a collection of quantities in stages, and also when we have output adjusted expectations to an expectation storage area which we want to be retained.

Beware that some controls force the unlocking of specified expectation stores. In particular, when the ae and scae controls are used to direct adjusted expectation output from an adjustment or a scan to an expectation store, the given expectation store is unlocked.

6.12 Generating beliefs automatically from sample covariances

▷ ▷ **Syntax**

1. BD>**vardata: v(S,B$_1$,B$_2$)** ↩

2. BD>**vardata: r(S,B$_1$,B$_2$)** ↩

where S *is a* belief store number *and* B_1, B_2 *are the names of* bases.

◁ ◁

The VARDATA: command is used to generate a variance-covariance matrix in a belief store from the sample variances and covariances calculated from data attached to the corresponding elements. In the first form of the syntax, sample covariances are calculated, and sample correlations are calculated for the second form of the syntax.

It is assumed that each of the elements comprising the bases B1 and B2 possesses data, and that there exists a subset of cases for which data is available for all of these elements. This subset defines the number of cases used to define the sample covariances (or correlations). Use of the correlation option requires that all the respective variances be positive. We use the multiplier $\frac{1}{n}$ rather than $\frac{1}{n-1}$ when determining these summary statistics. The beliefs generated are checked for symmetry (see §6.8.5).

6.13 Constructing new elements from old

▷ ▷ **Syntax**

1. BD>**build:** $\mathbf{N_0}$=[($\mathbf{E_1}$)][$\mathbf{N_1}$] ± [($\mathbf{E_2}$)][**B**] ± [($\mathbf{E_3}$)][**A**] ± ... ↩

2. BD>**build: A** ↩

where N_0, N_1 *are the names of* elements*,* $E_1, E_2, \ldots$ *are the names of* equations *(which must be enclosed in round brackets),* B *is the name of a* base*, and* A *is the name of an* assignment

◁ ◁

In the first form of the syntax, the BUILD: command is used to construct a new *element* from a linear combination of former *elements*, *assignments*, and *components*. In the second form of the syntax, the command is used to build a given *assignment*. The second form of the syntax is equivalent to issuing the first form of the syntax with

BD>**build: A=A** ↩

so that the assignment of the same name is constructed to be an element. Each construction takes place separately in every *unlocked* belief store. Expectations are only constructed in unlocked expectation stores. The LOCK: and ELOCK: commands may be used to change locks.

Base names may be included in the definition part to mean the sum of every element in the base, and where each element will take the coefficient given by the equation preceding the base name. For example, we might build the average of the base named B by issuing the command

BD>**build: average=(1/count(B))B** ↩

Each BUILD: command has a definition part which, after tracking through any assignments (possible recursively, and possibly taking taking into account any varying indices) will result in a linear combination of elements and components. Whenever there is a possible confusion between element names and assignment names, the element names have priority. To force the use of an assignment in case of doubt, the assignment name may be prefixed with the '!' symbol. Thus, for each name N in the definition part, the names of elements are checked firstly. If N is not an element, the assignment names are checked. If in addition N is not recognised as an assignment, it is assumed to be a component. The distinctions are that elements have beliefs explicitly associated with them; assignments are themselves lists of similar elements and components; and components may have functional beliefs specified over them. Note that an assignment is parsed recursively deeper until the definition part is a linear combination of either built elements or components, but never assignments. Actual construction of beliefs then takes place, searching for functional declarations if need be. The element being defined might already exist, in which case it will be overwritten. If the element already exists and appears in the definition part, the former definition is used fully as required, and then finally replaced after the conclusion of all calculations. Equations which have no succeeding element or assignment or component are treated as scalar parts of the linear combination, and are included in determining the expectation for the element.

Once any assignments have been traced through, the definition part for the new element N will be of the form $N = k + \alpha^T X + \beta^T C$, where k is some scalar, α is a vector of coefficients for the vector of elements X, and β is a vector of coefficients for the vector of components C. We may assume that X consists of all the elements defined to date, with corresponding $\alpha_i = 0$ if the element is excluded from the definition part. Beliefs over N are constructed as follows.

1. The expectation are found by

$$\mathrm{E}(N) = k + \alpha^T \mathrm{E}(X) + \beta^T \mathrm{E}(C),$$

   where all quantities are known except the vector $\mathrm{E}(C)$. $\mathrm{E}(C_i)$ is assumed to be zero unless an expectation has specifically been given for the component C_i by using the FE: command.

2. The variance is found by
$$\mathrm{Var}(N) = \alpha^T \mathrm{Var}(X)\alpha + \beta^T \mathrm{Var}(C)\beta + 2\alpha^T \mathrm{Cov}(X, C)\beta,$$
where $\mathrm{Var}(X)$ is a known matrix. The matrices $\mathrm{Cov}(X, C)$ (consisting of specifcations such as $\mathrm{Cov}(X_i, C_j)$) and $\mathrm{Var}(C)$ (consisting of specifications such as $\mathrm{Cov}(C_i, C_j)$) are assumed to have all their entries equal to zero *except* as specified by FVAR: commands relating the elements to components, and components to components.

3. The covariance between N and the other elements is found by
$$\mathrm{Cov}(N, X) = \alpha^T \mathrm{Var}(X) + \beta^T \mathrm{Cov}(C, X),$$
where $\mathrm{Var}(X)$ is known, and $\mathrm{Cov}(X, C)$ is addressed above. It is most important to understand that the BUILD: command is essentially memoryless: built elements do not remember their original definition parts. In particular, the component parts of the definition will not be taken into account in future use of the element. To illustrate the deficiency, suppose that you use the BUILD: command to construct firstly
$$N_1 = k + \alpha_1^T X + \beta_1^T C,$$
followed by
$$N_2 = k + \alpha_2^T X + \beta_2^T C.$$
This will result in the wrong value for $\mathrm{Cov}(N_1, N_2)$ being calculated as the term $\beta_1^T \mathrm{Var}(C)\beta_2$ will be absent. This term is absent because the linear combination used to construct N_1 has been forgotten; it is "remembered" only through the elements X, and not through the components C.

   To avoid this problem, the COBUILD: can be used to build elements from functional declarations, and to build up a large number of covariance structures together, but is generally considerably slower.

6.13.1 Example

A simple example involving assignments elements and components is as follows. Suppose that $B, C.1$ are element names, that D is an assignment including functional indices, and that V and W are the names of components summarising many like quantities. A number of declarations are shown in Figure 6.1. These are as follows:

1. The declaration of two elements, $B, C.1$ (the latter given an expectation of nine; the former having $\mathrm{E}(B) = 0$ by default) gathered into the base G, followed by their variance-covariance matrix.
2. Functional specifications for variances over the components V and W; covariances between V and $C.1$, and covariances between W and B. All other variance-covariance specifications (for example, between V and W) are to be zero by default.
3. A functional specification of an expectation for the component V.
4. The definition of the recursive assignment D, together with the definition of the element D_0 as a termination criterion for the recursive assignment.
5. The BUILD: command. When this command is activated, it will be discovered that $D.1$ is not an element, and so a check is made to see whether it is an assignment. It is, so $D.1$ is replaced by the linear combination it represents, and the linear combination to be built is thus $Y = 3 + B + C.1 + 5V.1 + W.1 + D.0$. Expectations, variances and covariances are now constructed for Y and between Y and $B, C.1$ from the explicit and implicit belief specifications given.

Now consider Figure 6.2. This is exactly the same sequence of commands as in Figure 6.1, except for the insertion of an extra command line building $D.1$ explicitly as the penultimate command line. This illustrates the second form of the syntax in that the assignment $D.1$ is taken, and the element $D.1 = C.1 + 5V.1 + W.1 + D.0$ constructed. It also illustrates the forgetfulness property addressed above in that the element Y constructed therafter will have different beliefs to the Y constructed in Figure 6.1, as the component constituents of the assignment $D.1$ will have been forgotten in the second case.

Figure 6.1: Constructing new quantities

```
BD>element: <G> B, C.1=9 ↩
BD>var: v(1,G) ↩
BD-2 ↩
BD-1 3 ↩
BD>fvar: v(1,V.i,V.j)=1 ↩
BD>fvar: v(1,W.i,W.j)=2 ↩
BD>fvar: v(1,V.i,C.j)=(.i=.j) ↩
BD>fvar: v(W.i,B)=.i-1 ↩
BD>fe: e(1,V.i)=5 ↩
BD>assign: D.k=C.k+(.k+4)V.k+W.k+D.(k-1) ↩
BD>element: D.0 ↩
BD>build: Y=B+(3)+D.1 ↩
```

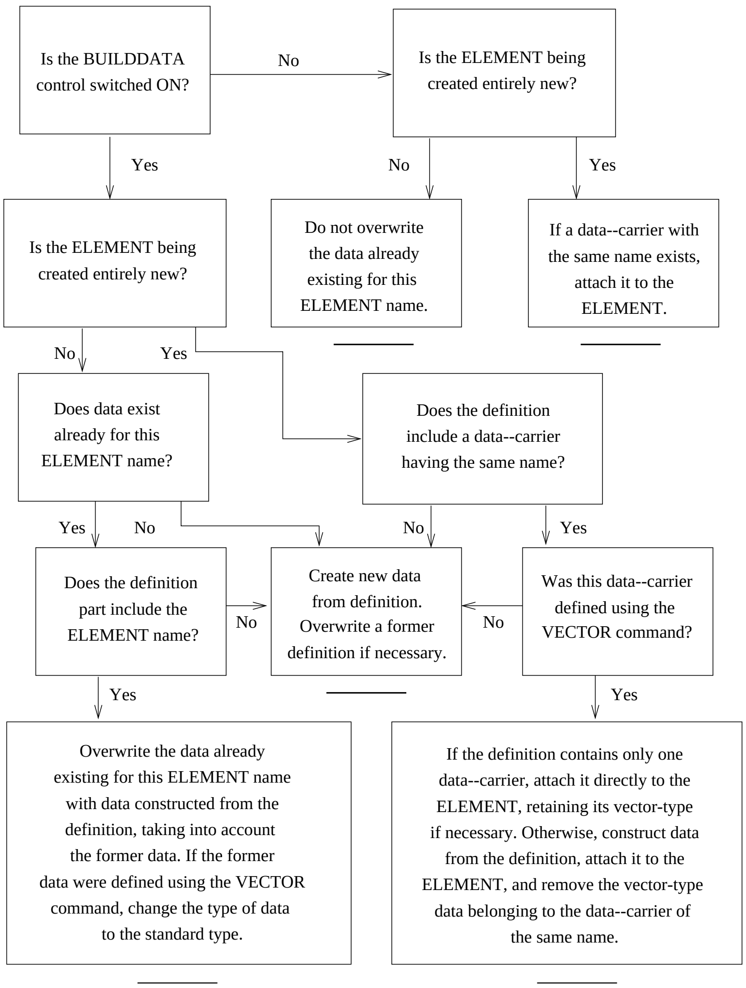
Figure 6.2: Illustrating forgetfulness

```
BD>element: <G> B, C.1=9 ↩
BD>var: v(1,G) ↩
BD-2 ↩
BD-1 3 ↩
BD>fvar: v(1,V.i,V.j)=1 ↩
BD>fvar: v(1,W.i,W.j)=2 ↩
BD>fvar: v(1,V.i,C.j)=(.i=.j) ↩
BD>fvar: v(W.i,B)=.i-1 ↩
BD>fe: e(1,V.i)=5 ↩
BD>assign: D.k=C.k+(.k+4)V.k+W.k+D.(k-1) ↩
BD>element: D.0 ↩
BD>build: D.1 ↩
BD>build: Y=B+(3)+D.1 ↩
```

6.13.2 Attaching data to the constructed element

If data exists on all elements in the definition part then data can be created for the new element. Data is constructed for, or attached to, the newly constructed element according to the scheme shown in Figure 6.3, which summarises both the issues involved and the actions taken in various circumstances. In general it is possible (1) to construct data from the elements and data carriers referred to in the definition part, and (2) if the element already existed beforehand, for the replacing element to adopt the data already attached beforehand. The default action is governed by the builddata control. The program handles any necessary conversions from data carriers defined using the FACTOR: command. Recall that components might also possess data, and this will be incorporated as appropriate.

Figure 6.3: Automatically attaching data to constructed elements



6.14 Generating many quantities together

6.14.1 Overview

It is possible to construct many elements simultaneously from general assignments such that the constructions do not suffer the forgetfulness property inherent in using the BUILD: command to contruct elements sequentially. As pointed out there, the components used to help construct elements for BUILD: commands are generally "forgotten" about once they have been used. With the COBUILD: command, the simultaneous construction means that the correct variances and covariances are constructed across the many elements built together. Generally, the COBUILD: command is used with assignments which include varying indices, and also with one-off components. The range of values for the indices actually to be used for the construction is set by either the INDEX: command or the INDEXVEC: command (depending upon whether the indices follow regular or irregular patterns respectively), and deleted by using the XINDEX: command.

6.14.2 Setting regular index ranges

▷ ▷ **Syntax**

BD>**index: C=E$_1$, E$_2$, E$_3$** $\hookleftarrow$

where C is an alphabetic character representing an index, and E_1, E_2, E_3 are equations.

◁ ◁

The syntax for the INDEX: command is similar to the syntax for the FOR: command, except that the arguments must be non-negative and the index C is a single alphabetic character. The command is used to vary the indices of quantities referenced in COBUILD: and ITADJUST: commands. An index can be overwritten - if an error occurs during re-definition, the former value is retained. The values used for indices should round to positive integers no larger than 32767. Index definitions can be deleted with the XINDEX: command.

6.14.3 Setting irregular index ranges

▷ ▷ **Syntax**

BD>**indexvec: C=D** $\hookleftarrow$

where C is an alphabetic character representing an index, and where D is the name of a data-carrier.

◁ ◁

The syntax for the INDEXVEC: command defines the name of an index, C, and a range of values to be associated with it. To use this syntax, it is necessary to have created a data-carrier D of length k, with k data cases $D(1), \ldots, D(k)$, beforehand. The intention is to replace the character C successively by the integers $round(D(1)), round(D(2)), \ldots, round(D(k))$.

The command is used to vary the indices of quantities referenced in COBUILD: and ITADJUST: commands. An index can be overwritten - if an error occurs during re-definition, the former value is retained. The values used for indices should round to positive integers no larger than 32767. Index definitions can be deleted with the XINDEX: command. When, for whatever reason, a data carrier is deleted or overwritten, and if it is also associated with an index, the index is deleted.

For example, suppose that you wish to construct (using the COBUILD: command) the elements a_{ijk} for $i = 1, 7, 8$, $j = 1, 2, 3, 4$, and $k = 2, 17, 19, 24$. Assuming that appropriate functional beliefs have been specified, the following code example shows how we could proceed.

```
index: j=1,1,4 ↩
data: <tempa>3 ↩
1 7 8 ↩
indexvec: i=tempa ↩
data: <tempb>4 ↩
2 17 19 24 ↩
indexvec: k=tempb ↩
cobuild: a.i.j.k ↩
```

The example shows the irregular patterns being stored as the data-carriers *tempa* and *tempb* respectively, and then associated with indices i, k respectively via the INDEXVEC: command. We define the index j via the INDEX: command as it is associated with a regular pattern.

6.14.4 Deleting index ranges

▷ ▷ **Syntax**

BD>**xindex:** C_1 [, C_2, ...] ↩

where $C_1, \ldots$ *are alphabetic characters representing indices.*

◁ ◁

The XINDEX: command is used to delete index ranges set by any preceding INDEX: and INDEXVEC: commands. The names of the indices to be deleted should be given as arguments. Any arguments which are not indices are ignored, and do not cause an error message to be reported.

6.14.5 Multivariate construction

▷ ▷ **Syntax**

BD>**cobuild:** $\mathbf{A}_1$ $\mathbf{IL}_1$ [, $\mathbf{A}_2$ $\mathbf{IL}_2$ ...] [, **F** $\mathbf{IL}_3$] ... ↩

where $A_1, A_2, \ldots$ *are the names of assignments,* $F, \ldots$ *are the names of components, and* $IL_1, IL_2, \ldots$ *are* index lists

◁ ◁

The COBUILD: command is the multivariate extension of the BUILD: command; it is used to build several realisations of quantities which have their relationships declared functionally via FVAR: commands and FE: commands, and so forth. The quantities in the definition part are either assignments or components. Components are treated as though they are assignments of the same name. The command should not be issued without a preceding INDEX: command as this sets the range of values for the index lists for which elements will be constructed.

6.14.6 Examples

Figure 6.4: Building many elements simultaneously

BD>**assign: Y.r.s=T+(.r)H.s.r+M.r** ↩
BD>**assign: N.r=(2)M.r** ↩
BD>**index: i=3,2,19** ↩
BD>**index: j=9,3,51** ↩
BD>**cobuild: Y.i.j, N.i** ↩

Figure 6.4 shows how the COBUILD: command might be used. It shows two assignments being made, which include quantities which we assume to be elements constructed beforehand or components. The INDEX: commands set the range of indices to be $i = 3, 5, \ldots, 17, 19$ and $j = 9, 12, \ldots, 51$. Finally, the COBUILD: command constructs simultaneously the elements $Y.3.9, N.3, Y.5.9, N.5, \ldots$, taking into account any explicitly defined beliefs about existing elements, and functionally defined beliefs expressed for components and between elements and components. A second example is shown in Figure 6.5.

Figure 6.5: Simultaneously constructing assignments and components

```
BD>assign: Y.t=(12)+(3)b.t+e.t ↩
BD>fvar: v(1,b.r,b.s)=(.r=.s)*12 ↩
BD>fvar: v(1,e.i,e.j)=1/(.i+.j-1) ↩
BD>fvar: v(1,e.i,g.j)=(.i=.j) ↩
BD>fvar: v(1,g.i,g.j)=(.i=.j)*57 ↩
BD>index: t=1,1,11 ↩
BD>cobuild: Y.t,G.t ↩
```

Chapter 7

Constants and functions

7.1 Introduction

By the term *constant* we mean a scalar value which is calculated once from its definition, and then remains available until it is redefined or deleted. By *function* we mean a scalar value which is calculated afresh from its definition every time it is required. In this way, functions serve as primitive subroutines. The C: command is used to define *constants*, and the F: command to define *functions*. They are deleted by using the XC: and XF: commands respectively. Constants and functions can be inspected by using the c and f arguments to the LOOK: command.

Generally, we precede the names of constants with the "%" symbol, and the names of functions with the "#" symbol. It is never a mistake to include these symbols, but they can sometimes safely be omitted, principally when they are being defined or deleted.

Constants and functions are used in [B/D] *equations* just as though they are numbers.

7.2 Defining constants

▷ ▷ **Syntax**

1. `BD>`**c: %C=E** ↩

2. `BD>`**c: %C=** ↩
`BD-`**E** ↩

where C is the name of the constant, and E is any valid equation.

◁ ◁

Constants have names, generally alphanumeric strings which begin with a letter, and are referenced by prefixing the name with a % symbol. Index notation is only allowed to the extent of numerical indices, and not indices representing unknowns (use *functions* instead). Hence, the name $\%x.2$ is allowed (meaning constant $\%x$ with known index 2), but the name $\%x.i$ is disallowed.

The first form of the syntax has the definition part following immediately after the name part. As an alternative, for interactive prompting of constant inputs, the second form of the syntax prompts for the definition part from the keyboard.

The definition part, E, is any valid *equation*. Thus, for example, E might consist of a simple real number, or some more complicated formula. Whichever, the result of the equation will be a single number and the constant will take this value until explicitly deleted or overwritten.

Constants can be deleted by using the XC: command, and they may also be redefined. It is an error if an unrecognised constant is found in an equation.

For an example, suppose that y is a data-carrier with observations $[y_1 \ldots y_{10}]$. The code shown in Figure 7.1 defines the constant %sum, initialising it to zero, and then accumulates the sum of the y_i's in the %sum constant. This sum is then output. Notice that the initialisation is strictly necessary.

Figure 7.1: Using constants

```
BD>c: %sum=0 ↩
BD>for: i=1,1,10 | c: %sum=%sum+y ( [i] ) ↩
BD>print: The sum of the Y values is (%sum) . ↩
```

7.3 Defining functions

▷ ▷ **Syntax**

1. BD>**f: #F=E** ↩

2. BD>**f: #F=** ↩
   BD***E** ↩

where F is the name of the function, and E is any valid equation*.*

◁ ◁

We define functions by using the F: command, which is used in parallel to the C: command. A function is the same as a constant except (i) that it is not evaluated until required; and (ii) may contain unknown indices. In contrast, a constant has a unique value which is unchanging. The value of a function changes according to its definition. If a function is defined solely in terms of numbers, it is equivalent to a constant.

Functions have names, generally alphanumeric strings which begin with a letter, and are referenced by prefixing the name with a # symbol. Function names typically terminate with one or more indices representing unknowns that will be revealed whenever the function is to be evaluated.

The first form of the syntax has the definition part following immediately after the name part. As an alternative, for interactive prompting of function inputs, the second form of the syntax prompts for the definition part from the keyboard.

The definition part, E, is any valid *equation.* Thus, for example, E might consist of a simple real number, or some more complicated formula. The equation may include unknowns, which must relate to the indices present in the function name. The value of these unknowns must be known at the time of evaluation of the function.

Functions can be deleted by using the XF: command, and they may also be redefined. It is an error if an unrecognised function is found in an equation.

Consider, for example, the definition and usage of a typical function shown in the code in Figure 7.2. This command line defines a function $\#X$ with unknowns represented by the indices .i, .j, and .k. Thus, for example, the intention will be to regard #X.1.2.3 as equivalent to the value of $(1 * 2) + 3^2 + \%sum$, where %sum is a constant (not necessarily already defined). The remarks made earlier about a function not being evaluated until required means that the value of the constant %sum will only be referenced at that time. Hence the first value output by this segment of code will be $\#X.1.1.1 = 1 + 1 + 1^2 + 13 = 16$, and the second will be $\#X.1.1.1 = 1 + 1 + 1^2 + 14 = 17$.

Figure 7.2: Using functions

```
BD>f: #X.i.j.k=(.i*.j)+sqr(.k)+%sum ↩
BD>c: %sum=13 ↩
BD>print: The current value of the function #X is (#x.1.1.1) . ↩
BD>c: %sum=14 ↩
BD>print: The current value of the function #X is (#x.1.1.1) . ↩
```

With regard to the attachment of indices to the function name to represent unknowns, the index is a period followed by an alphabetic character. This character must be replaced by an integer whenever the function is evaluated. When the formula part is evaluated to give the function its value, any character representing an unknown in the formula is replaced by its appropriate value; the context determines whether the period is also removed. Generally, the period is removed when the whole is intended to represent a number, and not removed when the whole is part of a name. For example, a definition of

BD>**f: #X.i.j= .i + .j + var(1,y.j.i) + %fred.i.j + z.i.j** ↩

would result in #X.3.4 being defined as $3 + 4 + v(1, y.4.3) + \%fred.3.4 + z.3(4)$, so that %fred.i.j is replaced by %fred.3.4, and not %fred34; whereas z.i(.j) is replaced by z.3(4). See section §6.3 for further information about functional forms.

7.4 Deleting constants

▹ ▹ **Syntax**

BD>**xc:** [%]**C1** [, [%]**C2,** ...] ↩

where $C1$, $C2$, ... , *are the names of* constants.

◃ ◃

We can use the XC: command to delete constants. The first form of the syntax deletes the constants whose names are $C1, C2, \ldots$. Names of constants which do not exist are ignored. The prefix symbol "%" associated with constants may be omitted.

7.5 Deleting functions

▹ ▹ **Syntax**

BD>**xf:** [#]**F1** [, [#]**F2,** ...] ↩

where $F1$, $F2$, ... , *are the names of* functions.

◃ ◃

We can use the XF: command to delete functions. The first form of the syntax deletes the functions whose names are $F1, F2, \ldots$, Names of functions which do not exist are ignored. Note in particular that only the alphanumeric part of the function name should be given – that is, *indices* representing unknowns should not be given. The prefix symbol "#" associated with functions may be omitted.

Chapter 8

Using data

8.1 Overview

A strict definition of *data* might be actual observations on random quantities. We extend this definition to mean any sequence or vector of real numbers, and the data handling facilities thus refer to these as well as to the actual observations. Two kinds of quantity are associated with data: *elements* and *data-elements*. The former are particularly associated with belief specifications, and some (or possibly all) elements may have no data associated with them. On the other hand, data-elements never have beliefs attached to them, but always are associated with data. We will tend not to distinguish between these two kinds of data-carrier.

Every data-carrier has a name, X for example, a number of cases $i = 1, 2, \ldots$, each of which can contain an observation, $X(i)$, a missing value marker, or a marker showing that no observation has yet been defined for this case. Different data-carriers can be – and often are – of different lengths. We sometimes refer to the entirety of observations on a data-carrier as a data vector.

The DATA: command is used to input multiple observations over collections of data quantities, and also to define single observations of a data quantity. The DATAVEC: command is used to define single values for a range of cases of a data quantity, and for fast construction of data vectors from other data vectors. The FACTOR: command is used to construct factors as data vectors, and to store them efficiently. Data are removed using the XDATA: command. Subsets of observations for various data-carriers can be defined for several applications using the SELECT: command.

Data can be reviewed in summary form by using the SUMMARY: command. Several features of the data (mean, variance, percentiles, etc.) are available via [B/D] operators and operands. See §21.10 for further details.

See §6.13, and especially the flowchart in Figure 6.3, for details on what happens when data is attached to *elements* which are the subject of a BUILD: command.

Data observations are initialised to -256, so the production of extraneous -256 values may be evidence of a bug. There is a limit (depending on machine implementation) to the number of observations which may be defined. See the program argument to the LOOK: command for further details.

8.1.1 Missing values

The missing value marker is -999 by default, although this may be changed to another number by using the missing control. This value is used to announce undefined cases to [B/D]; it is used with all three kinds of data definition: reading in multiple data cases as in §8.2, defining single observations as in §8.3, and the construction of data over many cases as in §8.4.

Any data value encountered which is equal to -999 (or the current missing value defined by the missing control to replace the default value of -999) represents an undefined case. [B/D] does not store the missing value, it simple records it as missing. Thus, the appearance of -999 in data lists output by [B/D] probably indicates a bug in [B/D].

[B/D] never stores more data values than it needs to, and does not retain data-elements whose data consists entirely of missing values. Thus, it is quite possible for data-elements to be deleted by assigning missing values to them. Here is an example. Suppose that we introduce the data-element called d with one observation, and then replace this observation by the missing value marker:

BD>**data: d(1)=5.8** ↩

BD>**data: d(1)=-999** ↩

This results in the deletion of the data-element d as finally it has no valid cases.

8.2 Reading data

▷ ▷ **Syntax**

1. BD>**data:** $<$**N**$_1$ **[,E**$_1$**] [,N**$_2$**] [,E**$_2$**] ...** $>$ **I @L(C)** ↩

2. BD>**data:** $<$**N**$_1$ **[,E**$_1$**] [,N**$_2$**] [,E**$_2$**] ...** $>$ **I @L** ↩

3. BD>**data:** $<$**N**$_1$ **[,E**$_1$**] [,N**$_2$**] [,E**$_2$**] ...** $>$ **I (C)** ↩

4. BD>**data:** $<$**N**$_1$ **[,E**$_1$**] [,N**$_2$**] [,E**$_2$**] ...** $>$ **I** ↩

where L is the name of a label *and* $0 \leq C \leq 6$ *is a valid* input channel number. *I is a positive integer representing the number of observations to be read in to [B/D].* $E1, E2, E3, \ldots$ *are valid* equations *which when rounded amount to a positive integer.* $N1, N2, \ldots$ *are the names of* bases, data-elements, *or* elements.

◁ ◁

If the first form of the command is issued, with no channel number being specified, the channel number is deduced - [B/D] does not allow duplicate labels. In the third form of the syntax, the beginning of the file associated with the input channel is indicated. In common with other macro facilities, the channel number can be zero (meaning the keyboard): a label should not be supplied in this case. The address implied by the channel and label supplied must exist at the time the command is issued. If the fourth form of the syntax is used, the data should appear on the lines following the command. If interactive data input is required, you may use either the third form of the syntax (interactively, or from a subroutine on an external file, using channel number 0), or by using the fourth form of the syntax.

8.2.1 Formatting the data for reading

Firstly we consider the order of data input. We envisage reading in a table of data, where each row corresponds to a different case, and each column to a different element or data-element. For smaller data sets, therefore, the most convenient data layout is as a table of numbers, with the numbers separated by one or more spaces.

However, the real numbers input need not necessarily be in tabular format They must be supplied in the order indicated, but you are free to split the input over several lines, as many numbers per line as you wish (possibly interspersed with blank lines) subject to the usual limitation of no more than 253 characters per physical line. A number must not be split over two or more lines. Any input on the same physical line as the last number required will be ignored.

Every N_i is the name either of a single quantity for which a column of data is to be read, or the name of a *base*. If N_i is the name of a base, the effect is as though N_i is replaced in the command line by all its constituents, in the order given by the ordering convention (see the glossary). Hence, the following two statements are equivalent from the point of view of inputting data:

1. BD>**base: B**$_1$**=N**$_2$**, N**$_1$ ↩
   BD>**data:** $< B_1 >$ 15 **@datahere (2)** ↩

2. BD>**data:** $<$**N**$_1$**, N**$_2$ $>$ **15 @datahere (2)** ↩

Every E_i should be (once rounded) a positive integer indicating a number of columns of data that should be skipped before the next column is input. There is no need to skip unwanted columns of data at the end a row.

8.2.2 Example

Consider the data shown in Table 8.1, consisting of four measurements on eight quantities. Suppose that we wish to read the second column into a data-carrier named X; the third and fourth columns into data-carriers called $A1$ and $A2$; and the seventh column into a data-carrier named Y. This could be achieved using the fragment of code shown in Figure 8.1, which we suppose to be part of a subroutine on an external file. (It is advisable not to input large quantities of data interactively: use a data file instead.)

Table 8.1: Formatting data for input to [B/D]

Time	P_1	P_2	P_3	P_4	P_5	P_6	P_7
3.85	74.10	74.26	75.67	74.09	73.10	73.58	69.24
3.95	74.10	74.25	75.66	74.08	73.09	73.58	69.24
4.05	74.10	74.25	75.65	74.06	73.08	73.57	69.24
4.15	74.10	74.24	75.64	74.05	73.07	73.57	69.24

Figure 8.1: Reading data into [B/D]

```
BD>base: A=A1,A2 ↩
BD>data: < 1, X, A, 2, Y > 4 ↩
BD>3.85 74.10 74.26 75.67 74.09 73.10 73.58 69.24 ↩
BD>3.95 74.10 74.25 75.66 74.08 73.09 73.58 69.24 ↩
BD>4.05 74.10 74.25 75.65 74.06 73.08 73.57 69.24 ↩
BD>4.15 74.10 74.24 75.64 74.05 73.07 73.57 69.24 ↩
```

8.3 Defining single observations

▷ ▷ **Syntax**

1. BD>**data: D(I)=$\mathbf{E_2}$** ↩

2. BD>**data: D($\mathbf{E_1}$)=$\mathbf{E_2}$** ↩

where D is the name of a data-carrier*, E_2 is a valid* equation *(optionally excluding parentheses), E_1 is a valid* equation *which should round to a positive integer (the rounding is performed by the program), and I is a positive integer.*

◁ ◁

In addition to reading lists of data for collections of quantities, the DATA: command can also be used to define one-off observations. Here, D is the name of some data-element or element (the data-element will be constructed if it does not exist already), I is the case being defined, and the result of the equation E will be assigned as the observation. For example, the following fragment of code

BD>**data: pressure(3)=50.7** ↩

makes the third observation on the data-carrier "pressure" equal to 50.7. The first form of the syntax is the most commonly used. The second form of the syntax allows the case number to be defined as an equation: the following two lines of code are equivalent:

1. BD>**c: %t=1** ↩
   BD>**data: x((%t+3))=5** ↩

2. BD>**data: x(4)=5** ↩

8.4 Constructing data over many cases

▷ ▷ **Syntax**

1. BD>**datavec: D=E** ↩

2. BD>**datavec: D(I₁,I₂)=E** ↩

where E is any valid equation*, and I_1 and I_2 are positive integers.*

◁ ◁

The DATAVEC: command is used for the speedy construction over many cases of new data from a definition which may be constant or involve other data. The command is best explained by example.

8.4.1 Introductory example

Suppose that E is a data-carrier having 20 observations $[E_1 \ldots E_{20}]$, and that you wish to construct a further data-carrier F with observations $F_i = 2E_i$. One valid way to do this is to use a FOR: loop in combination with an ordinary DATA: command:

BD>**for: i=1,1,20 | data: F([i])=2*E([i])** ↩

If instead we consider E and F to be data vectors, it is much more natural to consider writing something of the form $F = 2 * E$. The DATAVEC: command enables this form. That is, we replace the above line of code by the line

BD>**datavec: F=2*E** ↩

so that we don't refer explicitly to the individual cases. As another example, we could replace the line of code

BD>**for: i=1,1,20 | data: A[i]=B[i]+ln(C(2)/D[i]*E[i])** ↩

by the line of code

BD>**datavec: A=B+ln(C(2)/D*E)** ↩

where we drop the explicit case numbers. (Note that C(2) remains as it was, as the real number contained in C(2) is intended, rather than the data vector C). In practice, this way of constructing data vectors is also much quicker. We should observe various rules – to do largely with the selection of observations to be assumed – in applying the command as follows.

8.4.2 Constructing data from other data

The first form of the syntax for the DATAVEC: command is used mainly to construct data from other data. The definition part (the right hand side of the "=" sign) must include at least one data-carrier. The collection of data-carriers in the definition part is then reviewed and the cases for which there are observations on every member of the collection are selected. The definition part is then evaluated for each such case, and this becomes the "observation" for this case for the new data-carrier. As an example, suppose that X, Y, and Z are data-carriers with observations defined as in Table 8.2, and suppose that we construct a new data-carrier W as in the following fragment of code:

BD>**datavec: w=x+y/ln(z)** ↩

Table 8.2: Example data-carriers.

Case	Data-carrier X	Y	Z	Case selected? A	B	C
1	-	9.1	20	Yes	No	No
2	15.3	9.2	18	Yes	Yes	No
3	12.6	9.4	13	Yes	Yes	Yes
4	-	9.8	-	No	No	No
5	-	9.8	11	No	No	No
6	11.2	9.7	-	Yes	Yes	Yes

Only cases 2 and 3 exist for all three data-carriers in the definition part, and so observations will be constructed for these two cases only; the other cases remaining undefined. On the other hand, the issuing of the following command:

BD>**datavec: w=1** ↩

would result in an error as the definition part contains no data-carriers.

8.4.3 Constructing data for a range of cases

For the second form of the syntax, the range of cases for which observations are to be constructed are given explicitly as the pair of integers I_1, I_2. This form of the syntax must be used if there are no *data-carriers* in the definition part from which a selection of cases can be made. For example, to assign the same scalar to a data quantity for a range of cases, we would use a syntax similar to

BD>**datavec: X(1,20)=0** ↩

This second form of the syntax differs from the first form only in this regard: that an explicit range of values is given rather than deduced. Hence, if there are data quantities (data vectors) in the definition part, these are treated as in the first form of the syntax.

8.4.4 Generating vectors of random numbers

An important exception to the rule of scalars acting like scalars within DATAVEC: commands occurs when we include a random-number generator within the equation E. This is because the two random number generators **urand** and **nrand** supply different random numbers on each call, so that different values will be generated for the different cases. Consider, for example, the definition in this line of code:

BD>**datavec: X(1,100)=urand+1** ↩

This contructs a data vector X with observations $X(1)\ldots X(100)$ which will consist of a sequence of uniformly distributed (0,1) random numbers, plus 1. In contrast, consider the following fragment of code, which takes one random number and duplicates it over a range of cases:

BD>**c %temp=urand** ↩
BD>**datavec: X(1,100)=%temp** ↩

8.4.5 Remarks

The DATAVEC: command *overwrites* individual cases of previously existent data if a data-carrier with the same name already exists, or generates an entirely new data-element otherwise. For example, suppose that x does not yet exist as a data-carrier. The result of the following fragment of code:

BD>**datavec: x(1,4)=1** ↩
BD>**datavec: x(2,3)=2** ↩

is to create a data-element with observations as follows: $x(1) = 1$, $x(2) = 2$, $x(3) = 2$, and $x(4) = 1$.

8.5 Manual data selection

▷ ▷ **Syntax**

1. BD>**select: [TYPE] D [OP R]** [, ...] ↩

2. BD>**select:** ↩

where $TYPE$ is one of the three symbols (+,-,&); OP is one of the symbols (=, >, <, ≥, ≤, <>); R is a real number; and D is the name of a data-carrier

◁ ◁

The SELECT: command is used to select and deselect data cases. For most of [B/D]'s commands, the data selection is determined automatically to be the largest selection appropriate to the situation. However, it may sometimes be useful or necessary to take some subset of this largest possible selection, and thus enable the manual selection of data. This is achieved by judicious use of the SELECT: command, and by switching off the autoselect control.

We assume that prior to this command being issued there exists a prior selection, P. You might think of a selection as a notional set of flags attached to each observation number, where the flag is switched on if the observation number is to be selected, and off otherwise. The second form of the syntax sets the prior selection to be no cases.

Each usage of the SELECT: command consists of the following, repeated for as many data-carriers as you wish.

1. The name of a data-carrier, D.
2. Optionally, the $TYPE$ of selection. Where no $TYPE$ is given, the symbol '+' is assumed by default. The three $TYPE$s are as follows.

   - meaning that the selection is to become P *and not* D.

   $+$ meaning that the selection is to become P *or* D.

   & meaning that the selection is to become P *and* D.

3. A qualifier, consisting of an operator OP and a real number R, which restrict the number of cases to those cases of D which satisfy the qualifier. The possible OPs are

   $=$ meaning only those cases i for which $D_i = R$;

   $<>$ meaning only those cases i for which $D_i <> R$;

   $<$ meaning only those cases i for which $D_i < R$;

   $\leq$ meaning only those cases i for which $D_i \leq R$;

   $>$ meaning only those cases i for which $D_i > R$;

   $\geq$ meaning only those cases i for which $D_i \geq R$.

8.5.1 Examples

Suppose that we consider the scenario where we have three data-carriers, X, Y, and Z, whose observations are as in Table 8.2.

For our first example, suppose that there is no prior selection, and the following command is issued:

BD>**select: y<=9.7** ↩

This results in the selection of cases shown in column A in Table 8.2. Suppose next that the further command is issued:

BD>**select: &x** ↩

This results in the selection of cases shown in column B in Table 8.2, where a case is selected only if it appears both in the prior selection and exists for the data-carrier X. Suppose finally that this further command is issued:

BD>**select: -z=18** ↩

This results in the selection of cases shown in column C in Table 8.2, where the selection is as in column B except that the case corresponding to $Z = 18$ has been deselected.

Notice that we could have concatenated this sequence of commands as

BD>**select: y<=9.7, &x, -z=18** ↩

with exactly the same result.

8.6 Summarising data

▷ ▷ **Syntax**

1. BD>**summary: D1** [**,D2** ...] ↩

2. BD>**summary:** ↩

where $D_1, D_2, \ldots$ *are the names of* bases *or* data-carriers.

◁ ◁

The SUMMARY: command is used to summarise the data carried by elements and data-elements. For the first form of the syntax, a list of bases, elements, and data-elements is supplied. Data is summarised for every data-carrier in the collection formed by these quantities. (Each base is broken into its constituents; if any of these are bases then the process continues until all the constituents are data-carriers.) The second form of the syntax is used when we want to summarise all data. See §6.2 for details of specifying collections for the definition part.

Summaries are given only for *selected* data. Data may be selected in one of two ways. Firstly, the autoselect control may be switched on beforehand (its default). In this case, the data selection consists of all possible observations on all the quantities for which a summary is required, and thus the data selection is especially made for this particular SUMMARY: command. Secondly, the autoselect control may be switched off beforehand. In this case, the data selection consists of the current overall data selection, as made using the SELECT: command (see §8.5), and so the data selection should have been made prior to this particular SUMMARY: command. In either case, there might be missing values for some or all of the quantities.

The summary consists of, for each quantity for which a summary is required, the following values.

1. The number n of observations available for the current selection. Suppose these observations to be $[X_{(1)} \ \ldots \ X_{(n)}]$.
2. The smallest and largest of these n observations.
3. The average of these n observations, $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_{(i)}$.
4. The sample standard deviation of these n observations, $SD(X) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_{(i)} - \bar{X})^2}$.

Additionally, if the `scorr` and/or `scov` options are switched on, correlation and/or covariance output for all the pairs of quantities for which a summary is required. Correlation/covariance output is given provided that these additional circumstances are satisfied: (1) $n > 1$, (2) for every pair of quantities, an observation exists for the current selection.

As an example, consider the three data-carriers X, Y, Z and the three possible selections A, B, C shown in Table 8.2. Firstly suppose that the autoselect control is switched on. Then the current data selection is irrelevant, and a summary will be given which relates to three values on X, six values on Y, and four values on Z. No correlation output will be given as the number of observations summarised is different.

For selection A, the summary would relate to three values for X, four for Y, and four for Z. No correlation output would be available. Now suppose that selection B prevailed, and that we issued the command

BD>**summary: X, Y** ↩

Under this scenario, there are three selected matching observations: (15.3,9.2), (12.6, 9.4), and (11.2,9.7). Hence individual summaries for X and Y will be output, followed by a correlation based upon these three pairs if the `scorr` option is switched on, and a covariance matrix if the `scov` option is switched on.

8.7 Efficient usage of patterned data

▷ ▷ **Syntax**

BD>**factor: X=I$_1$,I$_2$,I$_3$** ↩

where X is the name of an element *or* data-carrier*, and I_1, I_2, I_3 are positive integers.*

◁ ◁

The FACTOR: command allows the construction of patterned data efficiently so that it occupies very little memory. X is the name of either of an *element* or *data-element*, or will be created as a new data-element. If data already exists for this named quantity, it is deleted. The effect of the command is to create observations for the cases $1 \ldots I3$, and for this range of cases, to define the observation for the i^{th} case as

$$X(i) = (((i-1)\, mod\, (i1 * i2))\; div\, i2) + 1$$

Thus the data vector consists of integers from $1 \ldots I1$, each integer being repeated consecutively I2 times in total; this process to continue until the specified length (I3) is reached. As an example, consider the following fragment of code:

BD>**factor: X=3,2,7** ↩

This results in the creation or overwriting of a data-carrier X having the 7 observations:

$$X(1) = 1, X(2) = 1, X(3) = 2, X(4) = 2, X(5) = 3, X(6) = 3, X(7) = 1.$$

Data created in this way uses no conventional data storage - it is evaluated and retained in functional form. However, if at any time data that was defined via a FACTOR: command becomes partly or wholly redefined, the efficient definition of the factor is deleted, and it is as though an entirely new data element is being defined.

8.8 Deleting data

▷ ▷ **Syntax**

BD>**xdata: D1** [**,D2** ...] ↩

where $D1, D2, \ldots$ are the names of data-carriers*, or* bases *containing data-carriers.*

◁ ◁

The XDATA: command is used to delete data. Data is deleted on the quantities given as the arguments following the command. The list of arguments can include bases. If so, all data-carriers included in the base have their data deleted. See §6.2 for details of specifying collections for the definition part.

Where all data on specifically a data-element is deleted, the data-element is also deleted. Otherwise, where the data-carrier is an *element*, only the data portion only is deleted.

Data defined using the FACTOR: command is also deleted.

If you delete a data carrier which has been associated with an index via the INDEXVEC: command, the index is also deleted. A warning is printed.

Chapter 9

Adjustment commands

9.1 Adjusting one collection by another

9.1.1 Overview

The ADJUST: command is the central tool of Bayes linear methodology as embodied in [B/D]. We focus on the three particular usages of the command. These are (1) the adjustment of one collection of elements by a number of others; (2) The partial adjustment of a collection by further collections, given others; and (3) The extraction of collections from an adjustment.

Adjustments may be performed with or without actual observations to hand: we do not distinguish between these two kinds of adjustment, although more aspects of the problem are examined when data is present. In the former case we also investigate the implications of our belief specifications, whilst the latter typically enable diagnostic comparisons between what was expected and what we observed.

[B/D] is designed to handle exchangeable situations. For these circumstances, the adjustments need to be organised in a particular way to exploit any exchangeability. Consequently we deal with this also below. In this chapter we will tend to refer to both elements and bases as information sources.

9.1.2 Preparing adjustments

A number of controls affect the kinds of adjustment that are carried out. Often the default values for these controls are acceptable, but for more advanced work you will need to consider altering them by using the CONTROL: command. Additionally, the adjustment should be organised so as to take account of any relevant data. Figure 9.1 charts the possibilities that need to be considered when preparing an adjustment.

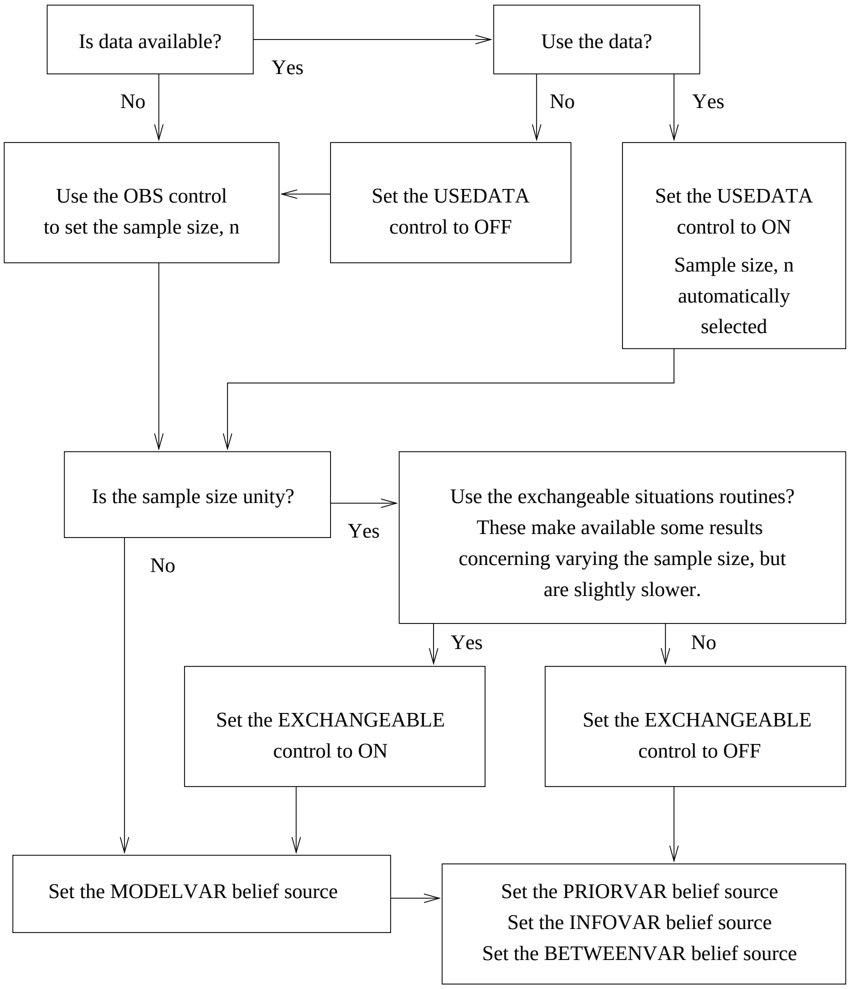
Organising data

An adjustment automatically takes account of any data present on the elements included in the collections to be used for the adjustment. For the observations to be included in the analysis, data must exist for every element included as an information source. If this is so, then the maximal number of cases for which there exist observations on each such element comprise the cases taken into account, and the number of cases is the sample size. If the sample size is greater than one, then you need to prepare for an exchangeable adjustment (see §6.6.3, §16.1). If the sample size under these rules turns out to be zero, then no data is available for the adjustment, and thus the results pertaining to the observed adjustment are not available. In these circumstances, [B/D] must be told what hypothetical sample size to assume. By default this is unity, but the obs control can be used to set any positive hypothetical sample size.

It might be that there is data available for every information source, but that you do not wish it taken into account. In this case the usedata control can be switched off, and the data (if any) are ignored. The obs control may be used, as above, to set a hypothetical sample size.

Occasionally you might wish [B/D] to operate upon some subset of the data. If so, you should switch off the autoselect control (which effects the automatic selection of data for many [B/D] commands) and then use the SELECT: command to select the desired subset of data.

Figure 9.1: Control settings for adjustments



Belief sourcing for non-exchangeable adjustments

We have pointed out elsewhere (see, for example, §6.6.3) that there may be alternative variance-covariance and expectation specifications for the same collection of elements, and these alternatives will be located in different belief and expectation stores. When we carry out an adjustment, we anticipate by default that all relevant variance-covariance specifications are in belief store number one, but this need not be the case. Indeed, for exchangeable adjustments this is often not the case.

As far as the expectation specifications are concerned, only the current default expectation store is ever referenced; the remainder are ignored. The default expectation store can be changed by using the e argument to the CONTROL: command.

Suppose that we gather all of the information sources $D_1, D_2, \ldots$ into the base D, and suppose that the sample size is unity. The information required by the adjustment is as follows:

- The expectations $\mathrm{E}(D)$ and $\mathrm{E}(B)$. These will have been specified earlier, when the elements were defined using the ELEMENT: command, and might since have been modified by using the E: command.
- Possibly, observations on D. If these are not available, no results for the observed adjustment are available.
- The prior variances specified over the collection being adjusted, $\mathrm{Var}(B)$. By default these variances are assumed to be present in belief store number one. If they are elsewhere, this can be changed by using the priorvar control as the adjustment assumes that the prior variance matrix over B can be found in the belief store pointed to by this priorvar control.
- The prior variances specified over the collection used for the adjustment, $\mathrm{Var}(D)$. By default these variances are assumed to be present in belief store number one. If they are elsewhere, this can be changed by using the infovar control as the adjustment assumes that the prior variance matrix over D can be found in the belief store pointed to by this infovar control.
- The prior covariances specified between the collection being adjusted and the collection used for the adjustment, $\mathrm{Cov}(B, D)$. By default these covariances are assumed to be present in belief store number one. If they are elsewhere, this can be changed by using the betweenvar control as the adjustment assumes that the prior covariance matrix between B and D can be found in the belief store pointed to by this betweenvar control.

Coherence and data consistency

The prior variances supplied, together with any observations over D, must be coherent. The COHERENCE: command can be used to perform coherence checking for variance matrices to be used in an adjustment. Otherwise, whenever an adjustment is calculated, [B/D] checks for the coherence of the variance specifications required for any analysis, and also checks that the data, if any, are consistent with the variance matrix specified over D, in the following context.

Suppose that the data quantities D for an adjustment are observed to be d. These data are inconsistent with the beliefs specified beforehand if, for any linear combination $q^T(D - \mathrm{E}(D))$, we have $\mathrm{Var}(q^T(D - \mathrm{E}(D))) = 0$ and $q^T(d - \mathrm{E}(D)) \neq 0$ (that is, we observed a change in a quantity whose value we believed we knew with certainty). The `datawarn` and `datawarn+` options are used to set the level of detail output when data inconsistency is detected, and the datawarn control can be set to warn about data inconsistency, rather than flagging it as an error. When flagged as an error, the default, the calculation of the adjustment fails. Otherwise, the adjustment proceeds as normal but of course the status of the results involving the data will be unclear.

Preparing exchangeable adjustments

For sample sizes greater than one, we must take account of the covariance structure between different observations on the collection D being used for the adjustment. These adjustments then exploit exchangeability. Even when the sample size is one, if you have available all the relevant beliefs, you can force the adjustment to be exchangeable by switching on the exchangeable control. You might do this, for example, if you are concerned with design issues involving a range of plausible sample sizes. In particular, output pertaining to the maximal resolution transform (crudely, what happens when you take an extremely large sample size) are available only for exchangeable adjustments. You may also wish to use special algorithms that are available for fast computation for pure exchangeable (but not general exchangeable) adjustments, that are enabled by switching on the matchcd control. (We recommend that you do so: see §16.3 for further details.)

We suppose that the quantities involved in the collection being used for the adjustment are of the form described in §16.1, that is a sequence of vectors $D_{(1)}, \ldots, D_{(n)}$. In the notation of that section, it remains to give the source of the belief specifications G and $G+U$. The latter is the variance matrix over each such vector $D_{(i)}$, and the former is the covariance matrix between a pair of vectors. That is,

$$
\begin{aligned}
G &= \mathrm{Cov}(D_{(i)}, D_{(j)}), \quad i \neq j \\
G+U &= \mathrm{Var}(D_{(i)})
\end{aligned}
$$

The infovar control should point to the belief store containing $G+U$, and the modelvar control should point to the belief store containing G. As above, the prior variances supplied, together with any observations over D, must be coherent. The COHERENCE: command can be used to perform coherence checking.

Preparing adjustments for influence diagrams

Influence diagrams are covered in greater detail in chapter 11. A diagram charting the various possibilities is shown in figure 9.2. In general, influence diagrams may be drawn when the `influence` option is switched on. The `influence` option permits the possibility of an influence diagram being drawn - you must still define the position of the nodes (representing the collections subject to the adjustment) by using the GRID: command, and specify connections between the nodes and so forth. Influence diagrams are not drawn when withdrawing quantities from an adjustment as in §9.1.5.

The overwrite control is used to determine whether or not the graphics screen is cleared at the end of each adjustment. If you need to construct an influence diagram which shows several related adjustments simultaneously, you will need to use this control. The overshade control is used to decide what happens to node shadings in the case of partial adjustments.

When influence diagrams are being drawn, rather more information than usual is required for the various node and arc label shadings. Consequently, adjustments for which we draw influence diagrams are rather slower than those that do not draw influence diagrams. Occasionally we might like to obtain the information used to draw the influence diagram without actually seeing it. In this case, the noinfluence control should be switched off, and the information will become available at the cost of slowing the program.

Abbreviated notation for bases and elements

It is often useful to be able to use abbreviations for the collections to be used for the adjustment. To do this we introduce a particular notation. Any base in the definition part of an ADJUST: or TESTGRID: command can appear as D, $< D >$, or $\{D\}$, with the following interpretation.

1. D - This is the simplest usage, and means the base D, is added as itself.

2. $< D >$ - This means that the base D is replaced by its constituent bases and elements, in alphabetical order. Thus if $D = [X2, C1, Z2, X1]$ where $X1, X2$ are elements and $C1, Z2$ are bases, then

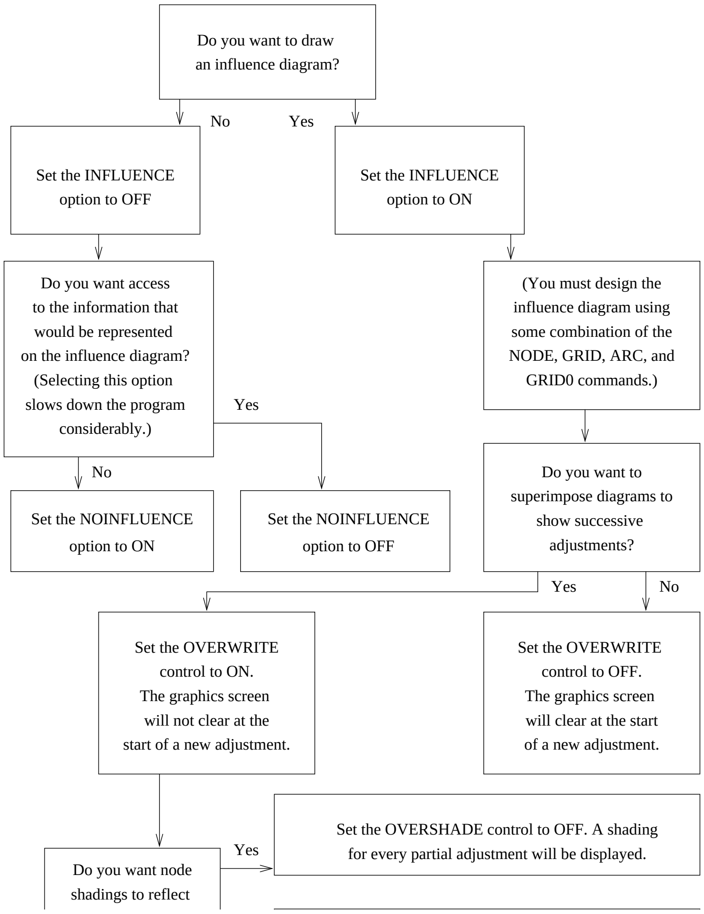
   BD>**adjust: [B / F+ < D > + G]** ↩

   is equivalent to

   BD>**adjust: [B / F+C1+X1+X2+Z2 +G]** ↩

3. $\{D\}$ - This means that the base D is replaced by its constituent *elements* in alphabetical order. Any bases contained in D are likewise replaced by their consituents and so forth, recursively, until only a collection of elements is left. For example, suppose that $D = [X2, C1, Z2, X1]$ where $X1, X2$ are elements and $C1, Z2$ are bases, where $C1$ contains the elements $A1, A2$ and $Z2$ contains the element $A0$. Then

Figure 9.2: Control settings for drawing influence diagrams for adjustments



BD>**adjust:** [**B** / **F**+ $\{D\}$ + **G**] ↩

is equivalent to

BD>**adjust:** [**B** / **F**+**A0**+**A1**+**A2**+**X1**+**X2** +**G**] ↩

Current and previous adjustments

Regarding adjustments, at any point the program may be in one of the following states. By an adjustment we mean the adjustment of some collection B by another D, and all the results pertaining to the adjustment - adjusted expectations, variances, resolution matrices, and so forth.

state 1 There is no adjustment defined.

state 2 The base to be adjusted is prepared for adjustment, but there is no adjustment as yet.

state 3 There is one single simple adjustment which we call the current adjustment.

state 4 A partial adjustment has been carried out following a simple adjustment. The overall adjustment is termed the current adjustment; the former adjustment is termed the previous adjustment; and results concerning the difference between the two (i.e. the partial adjustment) are available.

state 5 An *iterative* adjustment has been carried out. Under these circumstances, the current adjustment is not defined as these two kinds of adjustment are incompatible. See §9.3 for further details. A former iterative adjustment is cleared by an ADJUST: command.

Other remarks

- The name of the last information source to be used for the adjustment can be retained optionally using the fitted argument to the KEEP: command.
- The collection being adjusted can be retained optionally as a base by using the adjusted argument to the KEEP: command.
- Adjusted covariances as determined by an adjustment may optionally be retained in a belief store at the end of an adjustment. Use the ac control to achieve this, as described in §20.5.
- Adjusted expectations as determined by an adjustment may optionally be retained in an expectation store at the end of an adjustment. Use the ae control to achieve this, as described in §20.5.
- The rp control can be used to limit the degree of detail available for the resolution partition output, in particular the number of canonical quantities used to contribute to the resolution partition. See §20.13 for further details.

9.1.3 Simple adjustments

▷ ▷ **Syntax**

BD>**adjust:** [**B** / $\mathbf{D}_1$ + $\mathbf{D}_2$ + ... + $\mathbf{D}_r$] ↩

where B and $D_1, D_2, \ldots, D_r$ are the names of bases *or* elements.

◁ ◁

For simple adjustments we announce the base to be adjusted, B, and the collections to be used for the adjustment, $D_1, D_2, \ldots$. This leads to the following sequence of adjustments. Firstly, the base B is adjusted by the base D_1, giving the adjustment $[B/D_1]$. Secondly, the base B is adjusted by the base $D_1 \cup D_2$. As well as the standard adjustment, the partial adjustment is also reported. That is, we report both $[B/D_1 \cup D_2]$ and $[[B/D_1]/[D_2/D_1]]$, where the latter is the partial adjustment of B by D_2, having accounted for D_1. This process continues for the remaining information sources D_i: at each stage, we report the overall adjustment to that point, together with the partial adjustment representing the change between the i^{th} and the accumulation of the previous $(i-1)$ adjustments.

The base notation discussed in §9.1.2 may be used. Any base which is presented for use during an adjustment must contain only elements having beliefs, and no data elements.

9.1.4 Partial adjustments

▷ ▷ **Syntax**

1. BD>**adjust:** [**B** /] ↩

2. BD>**adjust:** [+ / $\mathbf{D}_1$ + $\mathbf{D}_2$ + ... + $\mathbf{D}_r$] ↩

where B and $D_1, D_2, \ldots, D_r$ are the names of bases *or* elements.

◁ ◁

The first form of the syntax defines and prepares the base B for adjustment, but does not carry out any adjustment. Thus, for example, the following two sequences of commands are equivalent:

1. BD>**adjust:** [**B** /] ↩
   BD>**adjust:** [+ / **N1** + **N2**] ↩

2. BD>**adjust:** [**B** / **N1** + **N2**] ↩

where $N1, N2$ are the names of bases. The SCAN: and ITADJUST: commands can be used similarly to force preparation for adjustment. The preparation is the same for any of these commands, so it doesnt matter which one you use. The base notation discussed in §9.1.2 may be used. Any base which is presented for use during an adjustment must contain only elements having beliefs, and no data elements.

The second form of the syntax is used for explicitly partial adjustments, and assumes that the base to be partially adjusted is defined already, either by the current adjustment, or by having prepared a base B for adjustment using the first form of the syntax. Suppose that the current adjustment (i.e. before this command is issued) is $[B/F]$. Then the following sequence of adjustments is carried out. Firstly, the base B is adjusted by the base $F \cup D_1$, giving the adjustment $[B/F \cup D_1]$. Secondly, the base B is adjusted by the base $F \cup D_1 \cup D_2$. At each stage both the overall adjustment and the partial adjustment is reported. That is, we report both $[B/F \cup D_1]$ and $[[B/F]/[D_1/F]]$, where the latter is the partial adjustment of B by D_1, having accounted for F. This process continues for the remaining information sources D_i: at each stage, we report the overall adjustment to that point, together with the corresponding partial adjustment representing the change between the new and former adjustments.

Regarding the data used for partial adjustments, subsequent partial adjustments following an initial adjustment must fit quantities such that they possess data consonant with the data selection used for the initial adjustment. Data averages are calculated afresh for *every* element in the supplied partial list, so it is possible to carry out the adjustment $[C/E]$, change the data on E, carry out the adjustment $[+/B]$ and so obtain different adjustments.

The preparation of adjustment is most useful when a sequence of adjustments is envisaged, as in the following example:

BD>**adjust: [B/]** ↩
BD>**for: i=1,1,10** ↩
BD>**adjust: [+/D[i]]** ↩
BD>**end:** ↩

9.1.5 Partial adjustments - withdrawing bases

▷ ▷ **Syntax**

BD>**adjust:** [- / $\mathbf{D}_1$ + $\mathbf{D}_2$ + ... + $\mathbf{D}_r$] ↩

where $D_1, D_2, \ldots, D_r$ are the names of bases *or* elements.

◁ ◁

In addition to adding quantities to a fit, we can withdraw them. This form of the command assumes that there is a current adjustment: suppose that it is $[B/F+D_1+\ldots+D_r]$ where B is the collection being adjusted; $D_1,\ldots,D_r$ are collections which have been fitted explicitly, and F comprises the remaining collections fitted. The effect of the command is now to carry out the adjustment $[B/F+D_2+\ldots+D_r]$ (dropping D_1), which becomes the new current adjustment. The difference between this and the previous adjustment is a partial adjustment which quantifies the effect of losing D_1 as an information source. This process continues, dropping a D_i at each stage until the final current adjustment is $[B/F]$.

The base notation discussed in §9.1.2 may be used. Any base which is presented for use during an adjustment must contain only elements having beliefs, and no data elements. Note that influence diagrams are not drawn when withdrawing quantities from an adjustment.

We emphasize that bases being withdrawn must have been explicitly added. That is, the following sequence of commands is not useful:

BD>**adjust: [B/D1+D2]** ↩
BD>**adjust: [-/D3]** ↩

as the base $D3$ would not have been explicitly fitted during the current adjustment, and so cannot be removed.

9.2 Displaying the results of an adjustment

9.2.1 Overview

There are huge numbers of results available for even the most trivial adjustment. These results can be displayed interactively by setting various output options using the OPTION: command. The various possibilities are described in some detail in §19.3. Every result that can be displayed interactively, as the adjustment is taking place, is retained and is available for display in exactly the same format at a later date by using the SHOW: command. This remains the case until the results of a new adjustment supercede the results of a former adjustment. It may be necessary or desirable to limit the output to a subset of the collection being adjusted. This is achieved by using the OPTION: command.

The great majority of results are also retained and available for use as further input. See, for example, §21.3 for standard adjustments, and in §21.5 for partial adjustments.

9.2.2 Displaying the results of an adjustment interactively

All of the results of an adjustment can be viewed as the adjustment takes places, or at a later date. To view a particular result as the adjustment takes place, the relevant output option must be set by using the OPTION: command described in chapter 19. The possible output options are described in §19.3. As an example, suppose that we are about to adjust the collection B by D, and that we wish to view the observed adjusted expectations and the adjusted variances as they are calculated. To achieve this, we issue the command

BD>**option: v+,a+** ↩

prior to issuing the ADJUST: command. These options remain in force until switched off, so you will see similar results for every subsequent adjustment. The `influence` option is used to display influence diagrams in a separate graphics window. The `pathsum` option should be used separately to the other text output options. It displays information which is intended to accumulate meaningfully over many adjustments. This information tends to become lost if other output options are selected, and so selection of the `pathsum` option switches off any competing options, and vice-versa.

9.2.3 Redisplaying the results of an adjustment

▹ ▹ **Syntax**

1. BD>**show: [Option1,Option2, ...] ([N1, N2, ...])** ↩

2. BD>**show: [Option1,Option2, ...]** ↩

where Option1, Option2, ... *is a list of output options separated by commas; and where* $N1$, $N2$, ... , *is a list of names of* elements *and* bases *in parenthesis and separated by commas.*

◁ ◁

The output options are those described in chapter 19. The list of names of elements and bases $N1$, $N2$, ... , defines a collection of elements to which output is to be restricted. Output is, in any case, only available for the elements in the collection being adjusted. See §6.2 for details of specifying collections for the definition part. The second form of the syntax gives output for all elements in the collection being adjusted.

The SHOW: command acts as the temporary version of the OPTION: command. That is, it delivers output for the specified options, and restricts the output to the subset defined by $N1$, $N2$, etc., but as soon as the requested output is delivered, the options and restrictions set previously by OPTION: commands come back into force.

A list of possible options is available by issuing the LOOK: command with argument options.

As pointed out in [37], various degeneracies imply that some [B/D] output is not unique, and [B/D] may not yield the same non-unique solution under replication.

9.3 Iterative adjustment

▷ ▷ **Syntax**

BD>**itadjust: [B / X IL]** ↩

where B is the name of a base *or* element, *X is the name of a* component, *and IL is an index list.*

◁ ◁

The ITADJUST: command uses an iterative adjustment algorithm, an alternative to our standard adjustment algorithm which increases the potential size of the fitting set, subject to certain limitations on the availability of some results. An example syntax is:

BD>**itadjust:[B / X.i.j.k]** ↩

where the intention is to adjust B by a number of quantities X_{ijk} for some range of values i, j, k.

The basic syntax is essentially the same, and implies the same, as the corresponding syntax for standard adjustments using the ADJUST: command. For example, these commands do redefine the current operator, and may also be used to define (or prepare) the collection being adjusted. Similarly, the collection being adjusted may have been prepared beforehand.

The second argument in the command consists of only one named quantity, X, with an index list. Notice that different quantities can be associated with different indices, if necessary. For example you might take $Z.i.1$ to represent $X.i$ and $Z.i.2$ to represent $Y.i$.

B is a base or single element which constitutes the collection to be adjusted, and which must carry beliefs that have already been specified. Beliefs specified between B and the quantities intended by X can be accessed or generated as follows:

1. Each element in B can be related directly to the X quantities via functional variance-covariance specifications using the FVAR: command.

2. Otherwise, if an element in B itself contains value indices (for example, if the first element in B is $Y.1.3$), then the program deduces whether the element plus its indices constitutes recognisable input to a FVAR: specification. For example, we might have defined

   BD>**fvar: v(2, Y.i.j, X.r.s.t) = E** ↩

   and this would be used to generate the requisite beliefs.

The second argument X represents the collection to be used for the adjustment. It is the name of a quantity whose beliefs have been specified functionally, using FVAR: and FE: commands, and which has an index list. The indices in the index list are given values by issuing an INDEX: command. For example, we might issue the following commands:

BD>**index:i=1,1,2** ↩
BD>**index:j=1,1,2** ↩
BD>**index:k=1,1,2** ↩
BD>**itadjust [B / X.i.j.k]** ↩

to mean that the base B should be adjusted by in turn by $X.1.1.1, X.1, 1, 2, \ldots, X.2.2.2$. Note that the order of adjustment is given by the final indices changing fastest. Functional beliefs should have been specified beforehand, although the index characters used in the definition need not be the same. For example, we might have made the definitions:

BD>**fvar: v(2, B.1, X.r.s.t)=E1** ↩
BD>**fvar: v(1, X.i.j.k, X.r.s.t)=E2** ↩
BD>**fvar: v(2, X.i.j.k, X.r.s.t)=E3** ↩
BD>**fe: e(1,X.j.k.r)=E4** ↩

where E1, E2, E3 and E4 are equations, usually containing varying indices, and $B.1$ is one of the elements in B. If we dont include functional specifications for the expectations for the quantities to be used in the adjustment, the expectations are taken to be zero.

As far as the expectation specifications are concerned, only the current default expectation store is ever referenced; the remainder are ignored. The default expectation store can be changed by using the e argument to the CONTROL: command. In the above piece of example code, expectation store number one has been used - this is the default expectation store number.

Belief store numbers are treated in the same way as for the standard algorithm, and so too are the various controls affecting adjustment which we discussed in §9.1.2. If a prospective sample size of greater than unity is used, then beliefs appropriate to an exchangeable adjustment must be available, and have been specified functionally. Location of the correct belief store is determined by the priorvar, betweenvar, infovar, and modelvar controls as for standard adjustments. The exchangeable control has no effect on the iterative algorithms, and commands which require varying the sample size are not available. Hence, the VARYSIZE: command is not available thereafter.

We use the ITADJUST: command where data is available, as follows. Using the same example, we need to set up the data system thus:

1. Arrange the data so that it exists on the data-carrier of the same name, X.
2. Introduce data carriers whose names are i, j, and k, to relate to the indices in the index list, $.i, .j, .k$.
3. Use these data-carriers (i,j,k) to define a selection over X as follows: $X(n)$ is selected if $i(n)$ equals the current value of index $.i$ AND if $j(n)$ matches the current value of index $.j$ AND if $k(n)$ matches the current value of index $.k$. The average of the selected data is taken as the datum for the element $X.i.j.k$ to be fitted.
4. The average for each element added must be based on the same sample size.
5. The FACTOR: command makes such data/index definition very simple.
6. Data selection is not governed by the usual selection facilities.

The output available is exactly the same as that under the standard ADJUST: command, and is obtained by setting the same options. In general, the iterative adjustment algorithm is slower than the standard adjustment algorithm, and production of the adjusted expectation is particularly time-consuming.

Some of the output for each successive iteration of the ITADJUST: command can be retained as data by setting various arguments to the KEEP: command. Results corresponding to consecutive iterations are stored in consecutive data locations. The results that can be retained are:

1. The expected sizes of consecutive adjustments, by using the itsize argument.
2. The observed sizes of consecutive adjustments, by using the itesize argument.
3. The path correlation between consecutive adjustments, by setting the itpath argument.

9.4 Assessing potential adjustments

▷ ▷ **Syntax**

1. BD>**scan:** [**B**/**F**] ↩

2. BD>**scan:** [+/**F**] ↩

3. BD>**scan:** [-/**F**] ↩

4. BD>**scan:** [**B**/] ↩

where B and F are the names of bases *or* elements

◁ ◁

We use the SCAN: command to quantify quickly a limited number of assessments of adjustments and partial adjustments. There are three possible scenarios, corresponding to the numbering in the syntax above: (1) scanning the affect of a basic adjustment; (2) scanning the effect of a partial adjustment; and (3) scanning the effect of a withdrawal of sources of information from an adjustment. The fourth form of the syntax, identical in effect to the equivalent ADJUST: form, is included for completeness only, and can be used to prepare a base for adjustment; such preparation is described in detail in §9.1.4. Generally, use of the SCAN: command does not affect the current operator (if any), and will not define one.

As far as expectation specifications are concerned, only the current default expectation store is ever referenced; the remainder are ignored. The default expectation store can be changed by using the e argument to the CONTROL: command.

9.4.1 Scanning the effect of a basic adjustment

For the first form of the syntax we evaluate limited aspects of the adjustment of the base B by the base F. The following information is available after such a command has been issued.

- the system adjustment uncertainty (essentially the standardised uncertainty remaining in the collection B having adjusted by F), is stored in the operand **scurvar**. In the notation of [31, page 13], the quantity stored is $\mathrm{U}_F(B)$.

- For each pair of elements in the base B, (B_i, B_j) say, we store their adjusted covariance, $\mathrm{Cov}_F(B_i, B_j)$. This then can be accessed via the [B/D] operator **scac** as **scac**(B_i, B_j) analagously to the [B/D] operator **ac**. For adjusted variances, we could also use the **scav** operator in that **scav**(B_i) is equivalent to, and shorthand for, **scac**(B_i, B_i)

- For each element B_i in the base B, we store the adjusted expectation, $\mathrm{E}_F(B_i)$. This then can be accessed via the [B/D] operator **scae** as **scae**(B_i) analagously to the [B/D] operator **aex**.

- By setting the scac control to some valid belief store, the potential variance-covariance matrix $\mathrm{Var}_F(B)$ may also be directed to a belief store. Every time the SCAN: command is issued, the adjusted covariances are then output to this belief store. This facility works independently of the availability of the adjusted covariances via the **scac** operator.

- By setting the scae control to some valid expectation store, the potential adjusted expectation vector $\mathrm{E}_F(B)$ may also be directed to an expectation store. Every time the SCAN: command is issued, the adjusted expectations are then output to this store. This facility works independently of the availability of the adjusted expectations via the **scae** operator.

Figure 9.3: A simple scan

```
BD>control: scac=3,scae=4 ↩
BD>scan: [B/F] ↩
BD>print Assessing the adjustment of base B by base F: ↩
BD>print The uncertainty remaining in the collection B is (scurvar) . ↩
BD>print The adjusted covariance between X and Y is (scac(X,Y)) , ↩
BD>print also available as (var(3,X,Y)) . ↩
BD>print The adjusted expectation for X is (scae(X)) , ↩
BD>print also available as (ex(3,X)) . ↩
```

As an example, consider the fragment of code given in Figure 9.3. Here, we scan for the effect of adjusting B by some other base F. Assume that the base B contains *elements* X, Y, and Z. To begin, we set the scac control equal to belief store 3, so that a copy of the adjusted covariances will be held in belief store 3 for all subsequent SCAN: commands. Similarly, we set the scae control equal to expectation store 4, so that a copy of the adjusted expectations will be held in expectation store 4 for all subsequent SCAN: commands. This is followed by the SCAN: command itself, and then various lines of output which show how we access the results of the command.

Similar information to this (albeit in terms of resolutions) can be obtained by using the **arcin** operator following an adjustment.

9.4.2 Scanning the effect of a partial adjustment

For the second form of the syntax we assume that an adjustment has already taken place. We will assume that the collection to be adjusted is the base B, and that the currently fitted collections can be arranged as the collection D. Hence, we assume the scenario that would result if we issued the command

```
BD>adjust: [ B / D ] ↩
```

It is also possible to use this second form of the syntax for the SCAN: command when a collection has been prepared for adjustment using either of the following two ways:

```
BD>adjust: [ B / ] ↩
BD>scan: [ B / ] ↩
```

in which case there is no current fit, and for the purposes of this section we will assume that D is empty. However, in such scenarios it would be more natural to use the first form of the syntax described above.

The action of the SCAN: command is now to evaluate limited aspects of the partial adjustment of the base B by the base F, having already adjusted by the base D. That is, the potential of the further adjustment by $[F/D]$ is assessed. The following information is available after such a command has been issued.

- the system adjustment uncertainty (essentially the standardised uncertainty remaining in the collection B having adjusted by both D and F), is stored in the operand **scurvar**. In the notation of [31, page 13], the quantity stored is $\mathrm{U}_{D\cup F}(B)$.
- For each pair of elements in the base B, (B_i, B_j) say, we store their adjusted covariance,

  $$\mathrm{Cov}_{D\cup F}(B_i, B_j).$$

  This then can be accessed via the [B/D] operator **scac** as **scac**(B_i, B_j) analagously to the [B/D] operator **ac**. For adjusted variances, we could also use the **scav** operator in that **scav**(B_i) is equivalent to, and shorthand for, **scac**(B_i, B_i)
- By setting the scac control to some valid belief store, the potential adjusted variance-covariance matrix $\mathrm{Var}_{D\cup F}(B)$ may also be directed to a belief store. Every time the SCAN: command is issued, the adjusted covariances are then output to this belief store. This facility works independently of the availability of the adjusted covariances via the **scac** operator.

- For each element B_i in the base B, we store the adjusted expectation,

$$\mathrm{E}_{D \cup F}(B_i).$$

  This then can be accessed via the [B/D] operator **scae** as **scae**(B_i) analagously to the [B/D] operator **aex**.

- By setting the scae control to some valid expectation store, the potential adjusted expectation vector $\mathrm{E}_{D \cup F}(B)$ may also be directed to an expectation store. Every time the SCAN: command is issued, the adjusted expectations are then output to this store. This facility works independently of the availability of the adjusted covariances via the **scae** operator.

As an example, consider the fragment of code given in Figure 9.4. Here, we scan for the effect of adjusting B by D and then some other base F. Assume that the base B contains *elements* X, Y, and Z. To begin, we switch off the scac control, so that we don't copy the adjusted covariances to a belief store for this and subsequent SCAN: commands. This is followed by the SCAN: command itself, and then various lines of output which show how we access the results of the command.

Figure 9.4: Scanning partial adjustments

```
BD>control: -scac ↩
BD>scan: [+/F] ↩
BD>print Assessing the partial adjustment by base F: ↩
BD>print The uncertainty remaining in the collection B is (scurvar) . ↩
BD>print The adjusted covariance between X and Y is (scac(X,Y)) . ↩
```

Similar information to this (albeit in terms of resolutions) can be obtained by using the **arcin** operator following an adjustment.

9.4.3 Scanning the effect of withdrawing information

For the third form of the syntax we assume that an adjustment has already taken place. We will assume that the collection to be adjusted is the base B, and that the currently fitted collections can be arranged as the (non-empty) collection D fitted explicitly as $D_1 + D_2 + \ldots + D_n$. Hence, we assume the scenario that would result if we issued the command

```
BD>adjust: [ B / D1 + D2 + ... + Dn ] ↩
```

The base scanned, F, must be one of the bases so added. That is, F must be $D1$ or $D2$ or ... or Dn.

The action of the SCAN: command is now to evaluate the partial loss in resolution that would result from withdrawing F from the adjustment.

For example, suppose that the base being withdrawn, F, is D_i. Then we assess the differences between the adjustments $[B/D]$ and $[B/\cup_{j \neq i} D_j]$.

The following information is available after such a command has been issued.

- the system adjustment uncertainty (essentially the standardised uncertainty remaining in the collection B having adjusted by D and then removing F), is stored in the operand **scurvar**. In the notation of [31, page 13], the quantity stored is

$$\mathrm{U}_{\cup_{j \neq i} D_j}(B).$$

- For each pair of elements in the base B, for example the pair (B_i, B_j), we store their adjusted covariance,

$$\mathrm{Cov}_{\cup_{j \neq i} D_j}(B_i, B_j).$$

  This then can be accessed via the [B/D] operator **scac** as **scac**(B_i, B_j) analagously to the [B/D] operator **ac**. For adjusted variances, we could also use the **scav** operator in that **scav**(B_i) is equivalent to, and shorthand for, **scac**(B_i, B_i)

- By setting the scac control to some valid belief store, the potential adjusted covariances

$$\mathrm{Cov}_{\cup_{j\neq i}D_j}(B_i, B_j)$$

may also be directed to a belief store. Every time the SCAN: command is issued, the adjusted covariances are then output to this belief store. This facility works independently of the availability of the adjusted covariances via the **scac** operator.

- For each element B_i in the base B, we store the adjusted expectation,

$$\mathrm{E}_{\cup_{j\neq i}D_j}(B_i).$$

This then can be accessed via the [B/D] operator **scae** as **scae**(B_i) analagously to the [B/D] operator **aex**.

- By setting the scae control to some valid expectation store, the potential adjusted expectations

$$\mathrm{E}_{\cup_{j\neq i}D_j}(B_i)$$

may also be directed to an expectation store. Every time the SCAN: command is issued, the adjusted expectations are then output to this store. This facility works independently of the availability of the adjusted expectations via the **scae** operator.

As an example, consider the fragment of code given in Figure 9.5. Here, we scan for the effect of removing F from the adjustment of B by D. Assume that the base B contains *elements* B_1, B_2, and B_3. The SCAN: command itself performs the partial withdrawal of information, and then there are various lines of output which show how we access the results of the command.

Figure 9.5: Scanning withdrawals of information

BD>**adjust [B/D1+D2+D3]** ↩
BD>**scan: [-/D2]** ↩
BD>**print Assessing the partial withdrawal of base D2:** ↩
BD>**print The uncertainty remaining in the collection B is (scurvar) .** ↩

Similar information to this (albeit in terms of resolutions) can be obtained by using the **arcout** operator following an adjustment.

9.5 Comparing alternative variance specifications

▷ ▷ **Syntax**

BD>**compare: B** ↩

where B is the name of any base *or* element.

◁ ◁

The COMPARE: command is used to compare two alternative variance-covariance specifications for a single collection of random quantities given by the base B. The compare control may be issued before the command is used (see §20.13) to set the sources for the alternative variance specifications.

As far as any expectation specifications are concerned, only the current default expectation store is ever referenced; the remainder are ignored. The default expectation store can be changed by using the e argument to the CONTROL: command.

Suppose that these sources are belief stores 1 and 2 respectively, and that we wish to compare $\mathrm{Var}_1(B)$ to $\mathrm{Var}_2(B)$. We require $\mathrm{r}(\mathrm{Var}_1(B) + \mathrm{Var}_2(B)) > 0$. Optionally, the output consists of a series of variance summaries and the corresponding directions. The variance summaries are output whenever the bcd option is switched on, and they are also output together with the corresponding canonical directions when the bcd+ option is switched on. The pairs representing the variance summaries are retained for future use as the paired operators **bcd1** and **bcd2**. The variance summaries are always reported and stored in ascending order, i.e. from smallest to largest eigenvalue.

- If the variance specifications share a common null space, there are eigenvectors v such that

$$\mathrm{Var}_1(v^T B) = \mathrm{Var}_2(v^T B) = 0.$$

These eigenvectors are output with a (0,0) pair representing the zero eigenvalues of each specification. The coefficients are normalised so that their squares sum to unity. The pair (0,0) is retained as **bcd1**$(i) = 0$ and **bcd2**$(i) = 0$.

- If the variance specifications are such that there exist eigenvectors v such that

$$\mathrm{Var}_1(v^T B) = 0, \quad \mathrm{Var}_2(v^T B) \neq 0,$$

then these eigenvectors are output normalised so that $\mathrm{Var}_2(v^T B) = 1$ together with a (0,1) pair representing their respective variances. The pair (0,1) is retained as **bcd1**$(i) = 0$ and **bcd2**$(i) = 1$.

- If the variance specifications are such that there exist eigenvectors v such that

$$\mathrm{Var}_2(v^T B) = 0, \quad \mathrm{Var}_1(v^T B) \neq 0,$$

then these eigenvectors are output normalised so that $\mathrm{Var}_1(v^T B) = 1$, together with a (1,0) pair representing their respective variances. The pair (1,0) is retained as **bcd1**$(i) = 1$ and **bcd2**$(i) = 0$.

- There remains a grid of directions for which the variances are positive under each specification. These directions are output, together with their variances and their ratio using the larger variance as a numerator. The eigenvectors are normalised so that each direction v has variance $\mathrm{Var}_2(v) = 1$. The pair of variance summaries $(\lambda_{(i1)}, \lambda_{(i2)})$ is retained as **bcd1**$(i) = \lambda_{(i1)}$ and **bcd2**$(i) = \lambda_{(i2)}$.

The directions and the variances reported may be interpreted as described in [27] and [34]. Note that the COMPARE: command may be used quite independently of any adjustment. Thus, the results pertaining to a previous adjustment remain available, and the previous adjustment remains defined.

The directions generated by the COMPARE: command may be retained as assignments by using the bcd argument to the KEEP: command.

Chapter 10

Miscellaneous graphics commands

10.1 Clearing graphics screens

▹ ▹ **Syntax**

BD>**gclear:** ↩

◃ ◃

The GCLEAR: command simply clears the graphics window.

10.2 Writing text to the graphics screen

▹ ▹ **Syntax**

BD>**gtext:** $\mathbf{R}_1$**,** $\mathbf{R}_2$**,** $\mathbf{I}_1$**,** $\mathbf{I}_2$**, T** ↩

where I_1 is an integer in $\{0,1,2,3\}$, I_2 is an integer representing a colour, R_1 and R_2 are real numbers in (0,1), and T is a string of alphanumeric text.

◃ ◃

The purpose of the GTEXT: command is to write the text string T to the graphics screen in a given size and colour. The size of the text ranges from $I_1 = 0$ (smallest) to $I_1 = 3$ (largest). The colour of the text depends upon the platform used: see Table 11.3 for a list of possible values. R_1 and R_2 are numbers in (0,1) which specify the (x,y) coordinates for the centre of the text. Notice that text can be 'deleted' on the graphics screen by redrawing it in the background colour, zero.

10.3 Printing high resolution graphics

10.3.1 Printing for the Windows version

The graphics window can only be output to the hardcopy device currently selected for Windows 3.1. This can be initialised or changed by running Print Manager from Control Panel in Windows 3.1. The hardcopy device can be altered to a file if required. To print the current graphics screen to the hardcopy device, click on the menu button at the top left of the graphics window, and select the **Print Window** option.

10.3.2 Printing for the UNIX version

The following refers to printing files using the UNIX version of [B/D] available in the Department of Mathematical Sciences, University of Durham. The graphics window can be output directly to a postscript file in the current directory by clicking on the button labelled **Postscript** at the top left of the graphics window. The filename is assigned automatically, and will be of the form **printoutn.ps**, where n is a positive integer. The saved postscript files do not overwrite previously existing files. Otherwise, the graphics window can be saved to a specific named file using the GPRINT: command as follows.

▹ ▹ **Syntax**

`BD>`**gprint: T** ↩

where T is any valid UNIX filename.

◃ ◃

The GPRINT: command is used to save the current graphics window to the postscript file given by T. It is an error if no file name is supplied. Any previously existing file of the same name will be overwritten. No checking is made that the filename supplied is valid.

Chapter 11

Influence diagrams

11.1 Introduction

[B/D] permits the construction of influence diagrams, using a graphics window, during ordinary use of the ADJUST: command. The influence diagrams drawn are as shown in [35], and follow very closely the schemata summarised in [32]. That is, nodes represent collections of quantities, and arcs drawn between two nodes represent a capacity of one to influence the other. Typically, we consider directed arcs, where the direction of influence is made explicit. By influence in this context we essentially mean the capacity of the information contained in one nodal collection to reduce uncertainty about another nodal collection. Our influence diagrams will frequently also display appropriate diagnostic information, where this is calculable.

The resolution for the UNIX and WINDOWS versions can be changed using the winxsize and winysize controls of §20.12.

11.2 Drawing influence diagrams during adjustments

We assume throughout this section that you have already used the GRID: command (or the GRID0: or NODE: and ARC: commands) to identify the nodes and arcs of interest. Suppose, for example, that you wish to adjust the collection B by the collections D, E and F, and so are about to issue the command

BD>**adjust: [B / D+E+F]** ↩

If we assume that you wish all these collections to appear on the influence diagram, then beforehand you should associate nodes on the diagram with these collections, and draw arcs between them as appropriate. The TESTGRID: command can be used quickly to show all the nodes and arcs relevant to the adjustment, without showing any other information.

A single option, the `influence` option, is used to require the display of the interactive construction of an influence diagram for the current adjustment. If the `influence` option is switched on, no other output is available interactively. However, results from an adjustment remain available (until overwritten by another adjustment) via later use of the SHOW: command. Figure 9.2 charts the different options available when drawing influence diagrams. The kind of shading used to shade the nodes to show summary and diagnostic information from an adjustment can relate either to the original quantities or to the canonical quantities. By default, shadings relate to the original quantities; otherwise the wheel control can be set to produce the *canonical wheel*.

We consider two kinds of influence diagram constructions: those resulting from a single adjustment, and those constructed from a sequence of adjustments. We discuss each of these as in the following examples. Influence diagrams can only be constructed under the fitting of sources of information; not the withdrawing of them.

11.2.1 Drawing one influence diagram per adjustment

Consider the example code in Figure 11.1, where A, B, D, E, F, X, Y, Z are the names of *elements* or *bases* to take part in the adjustment. The `influence` option is used to switch on the influence diagram display. Next, the overwrite control is switched off (it is switched on by default). When this command is switched off, the graphics screen is cleared when the new adjustment begins. The sequence of commands shown in Figure 11.1 will achieve the following:

1. display an influence diagram for the adjustment $[A/D + E + F]$;
2. clear the graphics screen;
3. display an influence diagram for the adjustment $[B/X + Y + Z]$.
4. the diagram will remain in the graphics window - you can return to the text window and resume processing.

Figure 11.1: Drawing single influence diagrams

BD>**option: influence** ↩
BD>**control: -overwrite** ↩
BD>**adjust: [A/D+E+F]** ↩
BD>**adjust: [B/X+Y+Z]** ↩

11.2.2 Using colours to track information

Each node is associated with a particular colour when the grid is defined via the GRID: command. Colours are used to make it possible to relate shadings representing uncertainty resolutions over nodes with the particular information sources used to remove the uncertainty. Suppose that the colour C (see Table 11.3 for a list of colours possible for different implementations) is associated with the node D, and that an arc leaves D and enters the node B. The effect of the colour is exhibited in three features of the influence diagram:

1. The outline for the node D itself is drawn in its colour C.
2. Any arc *leaving* the node is drawn in this colour, so the arc from D to B is drawn in colour C.
3. If we adjust B by D, the (possibly partial) resolution in B due to the information source D is shown by shading the appropriate proportion of the outer sector of B in the colour C. This allows us to examine shadings on a node and trace these directly back to the information source.

11.2.3 Superimposing influence diagrams

Now consider the example code in Figure 11.2, where A, B, D, E, F, X, Y, Z are the names of *elements* or *bases* to take part in the adjustment. The `influence` option is used to switch on the influence diagram display. Next, the overwrite control is switched on (its default). When this command is switched on, the graphics screen is not cleared after any adjustment command, and therefore any graphics drawn are superimposed over previously drawn screens. The GCLEAR: command can be used to clear the graphics output. The sequence of commands shown in Figure 11.2 will achieve the following:

1. display an influence diagram for the adjustment $[A/D + E + F]$;
2. superimpose an influence diagram for the adjustment $[B/X + Y + Z]$;
3. the diagram will remain in the graphics window - you can return to the text window and resume processing.

It is also possible to issue an ADJUST: command without naming the base to be adjusted, as in

BD>**adjust: [B/T]** ↩
BD>**adjust: [+/U+V]** ↩

In such cases, the effect as far as the influence diagram is concerned is exactly the same as if you had issued the command

Figure 11.2: Drawing superimposed influence diagrams

BD>**option: influence** ↩
BD>**control: overwrite** ↩
BD>**adjust: [A/D+E+F]** ↩
BD>**adjust: [B/X+Y+Z]** ↩
BD>**gclear:** ↩

BD>**adjust: [B/T+U+V]** ↩

The overshade control is used to determine whether or not the shadings for successive partial adjustments are redrawn or not, and only applies when the overwrite control is switched off. The ideas here are best illustrated by example. Suppose that you carry out the adjustment $[B/D_1 + D_2]$ and in doing so obtain an influence diagram where the B node is shaded first for the adjustment by D_1 and then by the partial adjustment given by fitting also on D_2. This gives a node with two contiguous shaded sectors, the whole representing the adjustment $[B/D]$ where $D = [D_1, D_2]$. Suppose that you now adjust partially by E and that you have switched off the overwrite control. This will result in another sector of the node being shaded, to correspond to the partial adjustment by E. However, the treatment of the shading of the former shaded sectors is as follows. If the overshade control is switched on, the sector corresponding to the adjustment by D is reshaded for that overall adjustment, thus replacing the initial two shaded sectors by one shaded sector. Otherwise the initial two shaded sectors are left as they were.

11.3 Arc influences and diagnostics

Influence diagrams contain arcs connecting nodes. We label the arcs as follows. We must first determine what kind of arcs, if any, are to be drawn. This is described in §11.4.1, with the different arc styles summarised in Table 11.4.

We will consider the case where the most detailed standard arcs might be drawn, corresponding to arc style number 7, and so we establish a scenario which might produce such details. The autoarc control can be used to draw particular arc types automatically between the nodes on an influence diagram, but we will concentrate on tailoring arcs to suit. Suppose that we adjust $[B/D + E + F]$, and produce an influence diagram with arcs connecting the information sources nodes D, E, and F with the node B.

We define the information flows from D, E, and F singly to B as $\mathrm{R}_D(B)$, $\mathrm{R}_E(B)$ and $\mathrm{R}_F(B)$ respectively, representing the worth of each information source in the absence of any other. The overall resolution at the node B is $\mathrm{R}_{D\cup E\cup F}(B)$, representing the total information arriving at B. If we observe $D = d, E = e, F = f$ then diagnostic quantities for these resolutions are given by the corresponding size ratios. Thus, for the arc between node D and node B, $\mathrm{Sr}_d(B)$ is the ratio given by dividing the observed maximal squared change in adjustment, $\mathrm{Size}_d(B)$, by its expectation, $\mathrm{R}_D(B)$.

We measure the information flow "from B to D" by the actual loss in resolution at B if node F is withdrawn from the adjustment. In our notation, this is $\mathrm{R}_{D\cup E}(B)$. We measure the information flow from B to E and from B to D similarly, by $\mathrm{R}_{D\cup F}(B)$ and $\mathrm{R}_{E\cup F}(B)$ respectively. Diagnostic quantities are also available for these measures; the diagnostic quantity corresponding to $\mathrm{R}_{E\cup F}(B)$ is $\mathrm{Sr}_{E\cup F}(B)$.

An assessment of how observed information sources work together (or against each other) is given by the path correlation which results when one information source is extracted. Hence, we can label the arc from node D to node B by $\mathrm{C}(d, e \cup f)(\mathrm{B})$, and the other nodes similarly.

The arc styles consist of (1) a straight line connecting two nodes; (2) a box containing information which is placed over the arc; and (3) a circle drawn towards the end of the arc, containing information about the path correlation. We describe the path correlation style first. A path correlation lies in $[-1, 1]$. We shade the circle according to the absolute magnitude of the correlation, so that correlations of 1 and -1 result in a fully shaded circle. We shade differently, according to direction. For positive correlations, we shade a corresponding proportion of the circle anti-clockwise, starting from 0 degrees. For negative correlations, we shade a corresponding proportion of the circle clockwise, starting from 0 degrees. The size of the circle can be changed using the pcradius control.

The box of labelling information consists of a bar divided into two. The half bar nearest the information source concerns information flow from the information source to the node being adjusted, i.e.

the contribution of the information source taken singly. The half bar nearest the node being adjusted concerns information flow away from the node being adjusted to the information source, i.e. the loss from extracting the information source singly.

Consider the arc drawn from D to B, where there is no data so that only influences are shown: Table 11.1 shows the contents of an arc label in this case. The region $A_{in} + B_{in} + C_{in}$ represents all the uncertainty in the destination node, proportionately 1. The region C_{in} represents the uncertainty remaining after all information sources have been fitted, $1 - \mathrm{R}_{D \cup E \cup F}(B)$. The region $A_{in} + B_{in}$ thus represents the proprtion of uncertainty removed in the destination node B. The region A_{in} represents the resolution in uncertainty in B due solely to fitting D, i.e. $\mathrm{R}_D(B)$. Thus, when A_{in} is large and B_{in} is small, the implication is that the single source of information D is virtually sufficient. When A_{in} is small and B_{in} is large, the implication instead is that the single source of information D is virtually useless. The outer half of the label is similarly configured: C_{out} is identical to C_{in}, and $A_{out} + B_{out}$ is identical to $A_{in} + B_{in}$. However, the region A_{out} represents the resolution in uncertainty in B due solely to extracting D from the adjustment, i.e. $\mathrm{R}_{E \cup F}(B)$. Thus, when A_{out} is large and B_{out} is small, the implication is that much of the resolution in uncertainty at B is lost if the single source of information D is withdrawn. When A_{in} is small and B_{in} is large, the implication instead is that the single source of information D contributes little extra to the information supplied already by $E \cup F$.

Table 11.1: Arc label contents (influences)

A_{in}	B_{in}	C_{in}	C_{out}	B_{out}	A_{out}
$\mathrm{R}_D(B)$	$\mathrm{R}_{D \cup E \cup F}(B) - \mathrm{R}_D(B)$	$1 - \mathrm{R}_{D \cup E \cup F}(B)$	$1 - \mathrm{R}_{D \cup E \cup F}(B)$	$\mathrm{R}_{D \cup E \cup F}(B) - \mathrm{R}_{E \cup F}(B)$	$\mathrm{R}_{E \cup F}(B)$

Now consider the case where we observe $D = d, E = e, F = f$, so that we can contrast actual to expected behaviour. The label that we draw is the same, except that we make a further partition of the regions A_{in} and A_{out} in Table 11.1. We obtain a label which resembles that shown in Table 11.2. We partition the region A_{in} according to the magnitude of the size ratio $\mathrm{Sr}_d(B)$. If this ratio is bigger than 1, we shade the portion $A^\star_{in}$, and otherwise we shade the portion A'_{in}. The actual proportions of A_{in} shaded depend on whether or not the ratio is larger than 1. If it is, the proportion of A_{in} shaded is $1 - (\mathrm{Size}_d(B))^{-\alpha}$, where $\alpha = 0.5$ by default, but can be changed using the bigshade control. If the ratio is smaller than 1, the proportion of A_{in} shaded is $1 - (\mathrm{Size}_d(B))^{\beta}$, where $\beta = 0.5$ by default, but can be changed using the smallshade control. The region A_{out} is partitioned similarly, except that the pertinent size ratio is $\mathrm{Sr}_{E \cup F}(B)$. The proportions shaded are as described for A_{in}, so that $A^\star_{out}$ is shaded if this ratio is larger than one, and A'_{out} is shaded otherwise.

Table 11.2: Arc label contents (influences and diagnostics)

$A^\star_{in}$	A'_{in}	B_{in}	C_{in}	C_{out}	B_{out}	A'_{out}	$A^\star_{out}$
$\mathrm{R}_D(B)$		$\mathrm{R}_{D \cup E \cup F}(B) - \mathrm{R}_D(B)$	$1 - \mathrm{R}_{D \cup E \cup F}(B)$	$1 - \mathrm{R}_{D \cup E \cup F}(B)$	$\mathrm{R}_{D \cup E \cup F}(B) - \mathrm{R}_{E \cup F}(B)$	$\mathrm{R}_{E \cup F}(B)$	

For both types of label, the labels may be modified by removing the unresolved proportions of uncertainty given by the C_{in} and C_{out} regions. This is achieved by adding 8 to the arc style. All other regions are then scaled up appropriately. This is useful when the actual resolution of uncertainty in the destination is very small, but when it is still important to understand information flow.

It is also possible to draw labels which emphasize the diagnostic information carried by the arc. This is achieved by adding 16 to the arc style. The effect is to obtain labels like those shown in Table 11.2, but with the regions B_{in}, C_{in}, C_{out}, and B_{out}, removed and the remaining regions scaled up.

The length and width of arc labels are set by default, but may be changed by using the arclength and arcwidth controls.

11.3.1 Accessing arc labelling information

A number of the actual values used to label arcs are available directly as [B/D] operators as follows.

- The **arcin** operator can be used to access the resolutions shaded in region A_{in} of Table 11.1.
- The **arcout** operator can be used to access the resolutions shaded in region A_{out} of Table 11.1.
- If appropriate, the **arcind** operator can be used to access the size ratios shaded in region A_{in} of Table 11.2. The actual size ratio is returned.
- If appropriate, the **arcoutd** operator can be used to access the size ratios shaded in region A_{out} of Table 11.2. The actual size ratio is returned.
- If appropriate, the **arcpc** operator can be used to access the path correlation drawn on an arc.

These operators are always available when an influence diagram has been drawn. However, as it is costly to calculate the measurements needed for the labels, by default they are not calculated for standard adjustments. However, you can force calculation of these values by switching off the noinfluence control. The SCAN: command returns similar labelling information, although in terms of overall adjusted uncertainties rather than partial resolutions.

11.4 Designing and testing influence diagrams

11.4.1 Designing the influence diagram

▹ ▹ **Syntax**

1. BD>**grid:** ↩
BD-$\mathbf{B}_1$,$\mathbf{B}_2$, ... , $\mathbf{B}_n$ ↩
BD*$\mathbf{X}_1$,$\mathbf{X}_2$, ... , $\mathbf{X}_n$ ↩
BD*$\mathbf{Y}_1$,$\mathbf{Y}_2$, ... , $\mathbf{Y}_n$ ↩
BD*$\mathbf{R}_1$,$\mathbf{R}_2$, ... , $\mathbf{R}_n$ ↩
BD*$\mathbf{C}_1$,$\mathbf{C}_2$, ... , $\mathbf{C}_n$ ↩
BD*$\mathbf{U}_1$,$\mathbf{U}_2$, ... , $\mathbf{U}_n$ ↩
BD*$\mathbf{V}_1$,$\mathbf{V}_2$, ... , $\mathbf{V}_n$ ↩
BD*$\mathbf{A}_{11}$,$\mathbf{A}_{12}$, ... , $\mathbf{A}_{1n}$ ↩
BD*$\mathbf{A}_{21}$,$\mathbf{A}_{22}$, ... , $\mathbf{A}_{2n}$ ↩
BD* ⋮ ↩
BD*$\mathbf{A}_{n1}$,$\mathbf{A}_{n2}$, ... , $\mathbf{A}_{nn}$ ↩

2. BD>**grid:** [**@L** [**(C)**]] ↩

3. BD>**grid0:** ↩
BD-$\mathbf{B}_1$,$\mathbf{B}_2$, ... , $\mathbf{B}_n$ ↩
BD*$\mathbf{X}_1$,$\mathbf{X}_2$, ... , $\mathbf{X}_n$ ↩
BD*$\mathbf{Y}_1$,$\mathbf{Y}_2$, ... , $\mathbf{Y}_n$ ↩
BD*$\mathbf{R}_1$,$\mathbf{R}_2$, ... , $\mathbf{R}_n$ ↩
BD*$\mathbf{C}_1$,$\mathbf{C}_2$, ... , $\mathbf{C}_n$ ↩
BD*$\mathbf{U}_1$,$\mathbf{U}_2$, ... , $\mathbf{U}_n$ ↩
BD*$\mathbf{V}_1$,$\mathbf{V}_2$, ... , $\mathbf{V}_n$ ↩

4. BD>**grid0:** [**@L** [**(C)**]] ↩

where $[B_1 \ldots B_n]$ *are the names of* n bases *or* elements*;* $[X_1 \ldots X_n]$, $[Y_1 \ldots Y_n]$, $[R_1 \ldots R_n]$, $[U_1 \ldots U_n]$, *and* $[V_1 \ldots V_n]$ *are real numbers;* $[C_1 \ldots C_n]$ *are integers in [1,15]; and* $[A_{11} \ldots A_{nn}]$ *are integers. L is the name of a* label*, and C is a valid input channel number: an integer such that* $1 \leq C \leq 6$.

◁ ◁

The GRID: command is used to define various basic features for the quantities to appear on an influence diagram. Once used, its essential features can be examined by issuing the LOOK: command with argument grid, and the TESTGRID: command can be used to test the design. The command typically is spread over $n+8$ [B/D] command lines. For the second form of the command an address is given, and the remaining $n+7$ lines should then be placed following the given address. The inputs to the command are as follows.

Line 2: defines a list of *bases* (or *elements* in their role as bases containing only one element) which are to be viewed as nodes. If the name of any base or element given here is already defined as a node, the former definition is overwritten and any arcs connecting this node to any other are removed before the new arcs are defined.

Lines 3 and 4: We consider the coordinate system for the influence diagram as a unit square, with bottom left coordinate (0,0) and top right coordinate (1,1). A location for the centre of each node in this square should be given on lines 3 and 4, such that the centre of node B_i locates at (X_i, Y_i).

Line 5: We give the radius of every node to appear on the diagram. That is, R_i will be the radius of the node associated with the base B_i. It is possible to indicate here, by setting $R_i = 0$, that [B/D] should use it's default radius.

Line 6: We give the colour to be associated with every node to appear on the diagram. That is, C_i will be the colour attached to the node associated with the base B_i. Each value of C_i is an integer in [1,15], representing a colour. A table showing the colours represented by these integers is shown in Table 11.3. The colour give for the node determines (1) the colour used to draw the node; (2) the colour of any arc leaving the node; (3) the colour used to shade the outer sectors of any nodes to which there is a directed arc connected to this node, and which represent collections which are to be adjusted by this node.

Table 11.3: Colours available for graphics images

	Monochrome	4 Colours (UNIX)	8 Colours (Deskjet 500C)	16 Colours (EGA/VGA)
1	black	black	black	darkgray
2	black	red	red	lightred
3	black	blue	blue	lightblue
4	black	green	green	green
5	black	red	magenta	lightmagenta
6	black	blue	cyan	lightcyan
7	black	green	yellow	yellow
8	black	red	green	lightgreen
9	black	blue	red	red
10	black	green	blue	blue
11	black	red	magenta	magenta
12	black	blue	cyan	brown
13	black	green	yellow	darkgray
14	black	red	green	lightgray
15	black	blue	red	cyan

Lines 7 and 8: We label the nodes by their base or element names. In the same way that we locate the centre of the node, we locate the centre of the node label in the unit square. That is, the centre of the label to be attached to node B_i will be located at (U_i, V_i).

Lines 9 to $n + 8$: The remaining n lines are used to define the arcs and their directions. Each A_{ij} is an integer between 0 and 31 inclusive which will set the type of the arc drawn from node B_i to node B_j. The possible values and their meanings are shown in Table 11.4. An arc can consist of (1) a path correlation, if appropriate; (2) a label summarising arc influence; and (3) the arc itself: a directed straight line connecting two nodes. In addition, two further styles may be selected: (4) the arc information can be made to be proportional either to the actual resolution at a destination node, or to the overall uncertainty at the destination node; and (5) arc diagnostic information only might be shown. These possibiliies may be selected separately. An integer in $0 \ldots 31$ is used to set the arc type. The binary representation of this integer is used to determine what possibilities are selected. The integer has a 5 bit binary representation, with a bit equal to 1 if a style is selected, and 0 otherwise. For example, arc style 19 (10011 binary) has the first, second, and fifth style bits set, and consists of a straight line and a label, and the labelling information is primarily diagnostic.

Table 11.4: Arc styles

Bit	Bit value	Interpretation
1	0	No arc line drawn
	1	Arc line drawn
2	0	No label drawn
	1	Label drawn
3	0	No path correlation label drawn
	1	Path correlation label drawn
4	0	Arc label information proportional to overall nodal uncertainty
	1	Arc label information proportional to overall nodal resolution
5	0	Arc diagnostic information not to cover the entire label.
	1	Arc diagnostic information to cover the entire label.

Notice that we would define $A_{ji} = 1$ and $A_{ij} = 0$ to draw an arc from node B_j to node B_i. If $A_{ij} = 3$ then, if appropriate, the information concerning arc influence and a diagnostic quantity will be drawn in the form of a bar overlaying part of the arc.

The real numbers input need not necessarily be in the format suggested. Suppose that there are n nodes being defined. The numeric information following the GRID: and GRID0: commands consists of $n(n + 6)$ and $6n$ real numbers respectively. These numbers must be supplied in the order indicated, but you are free to split the input over several lines, as many numbers per line as you wish (possibly interspersed with blank lines) subject to the usual limitation of no more than 253 characters per physical line. A number must not be split over two or more lines. Any input on the same physical line as the last number required will be ignored.

Consider the code given in Figure 11.3 for a fictitious example. This defines three nodes for collections named *intelligence, environment*, and *heredity* respectively. The nodes are centred at (0.3,0.5), (0.7,0.7), and (0.7,0.3) respectively; and their names are placed respectively underneath, above, and underneath the nodes. Each node will be drawn with a radius equal to 0.04. There are only two directed arcs, one from the environment node to the intelligence node; and the other connecting the heredity node to the intelligence node. The resulting diagram should resemble Figure 11.4.

The GRID0: command is identical to the GRID: command except that no arcs are defined. This is useful when designing the partial correlation diagram, as arcs connecting the nodes on such diagrams are generated according to various other criteria. Notice that the ARC: command can be used to specify arc connections piecemeal.

11.4.2 Testing the design overall

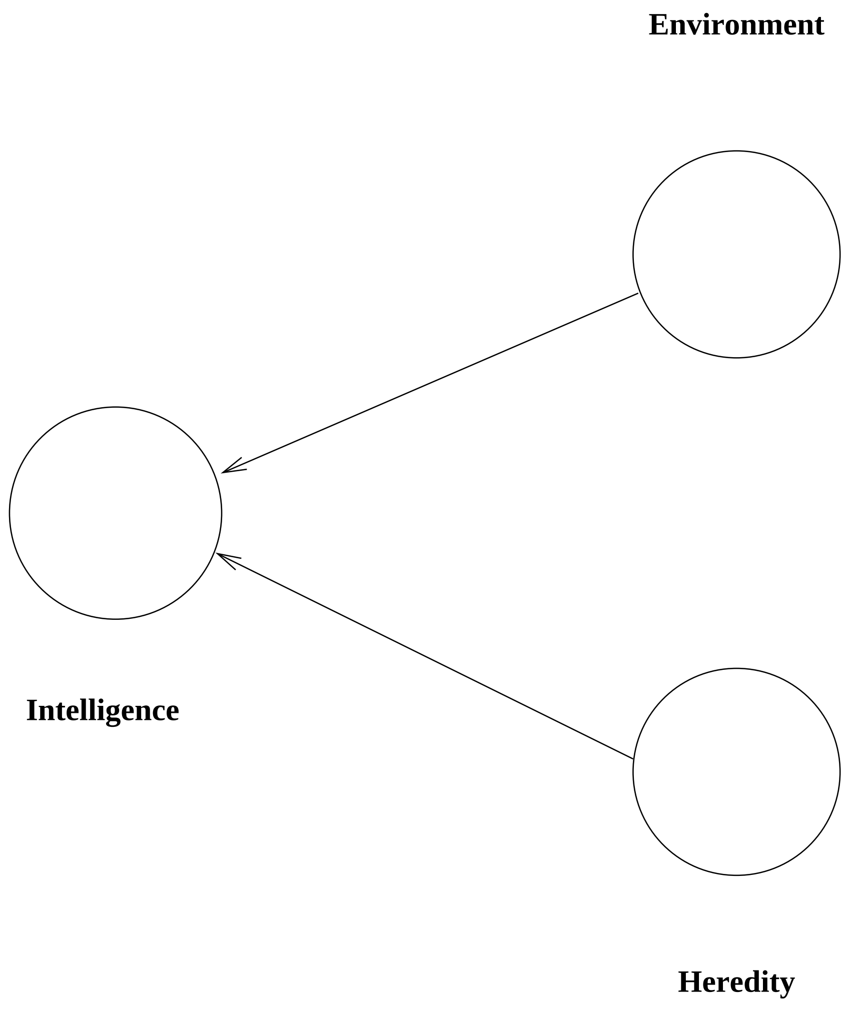
▷ ▷ **Syntax**

1. `BD>`**testgrid:** ↩

Figure 11.3: Example influence diagram design

```
BD>grid: ↩
BD-intelligence,environment,heredity ↩
BD*0.3 0.7 0.7 ↩
BD*0.5 0.7 0.3 ↩
BD*0.04 0.04 0.04 ↩
BD*2 3 4 ↩
BD*0.3 0.7 0.7 ↩
BD*0.4 0.8 0.2 ↩
BD*0 0 0 ↩
BD*1 0 0 ↩
BD*1 0 0 ↩
```

Figure 11.4: A crude influence diagram



2. BD>**testgrid: [B / $\mathbf{D}_1$+$\mathbf{D}_2$+ ... + $\mathbf{D}_n$]** ↩

where B and [D_1 ... D_n] are the names of elements *and* bases.

◁ ◁

To help organise the influence diagram, the TESTGRID: command issued with no arguments draws all the nodes and arcs defined to date. The drawing will be superimposed over the last drawn graphics screen unless the overwrite control is switched off.

The second form of the syntax is used to gauge the appearance of an influence diagram that would result from a future ADJUST: command issued with the same parameters. The names of the bases and elements B, [D_1 ... D_n], may or may not have been associated with nodes via a GRID: command beforehand - if not, they will not appear on the diagram. As before, the drawing will be superimposed over the last drawn graphics screen unless the overwrite control is switched off.

Abbreviated notation for the bases and elements in the definition part of the command may be used as described in §9.1.2.

11.4.3 Defining specific nodes

▷ ▷ **Syntax**

BD>**node: B,$\mathbf{R}_1$,$\mathbf{R}_2$,$\mathbf{R}_3$,I,$\mathbf{R}_4$,$\mathbf{R}_5$** ↩

where B is the name of an element *or* base. *I is an integer, and $R_1, R_2, \ldots, R_5$ are real numbers in (0,1).*

◁ ◁

The NODE: command is used to specify nodal information for a single node to be drawn on an influence diagram. The information supplied is as for a GRID: command, and in the same order. That is,

- B is the name of the base or element to be represented as a node on the influence diagram.
- R_1, R_2 are numbers in (0,1) which specify the (x,y) coordinates for the centre of the node.
- R_3 is a number in (0,1) gives the radius of the circle representing the node.
- I is an integer in [1,15] representing a colour (see Table 11.3).
- R_4, R_5 are numbers in (0,1) which specify the (x,y) coordinates for the label associated with the node.

Further details can be found under discussion of the GRID: command. The NODE: command should be treated simply as an alternative syntax for the GRID0: command when only one base or element is being defined as a node.

11.4.4 Defining specific arcs

▷ ▷ **Syntax**

BD>**arc: $\mathbf{B}_1$, $\mathbf{B}_2$, A** ↩

where B_1 and B_2 are the names of elements *or* bases. *A is an integer.*

◁ ◁

The ARC: command is used to specify or redefine arcs between two given nodes B_1 and B_2 on an influence diagram. The arc type is given by the integer A, which should be an integer constructed from the bits discussed in Table 11.4. Any pre-existing definition of an arc between B_1 and B_2 is overwritten. In particular, if $A = 0$, meaning no arc between B_1 and B_2 then the former arc definition, if any, is deleted.

The ARC: command can be used in combination with the GRID0: command to generate influence diagrams with a small number of arcs. The autoarc control can be used to have [B/D] draw arcs automatically. See §20.9 for details.

11.5 Drawing arcs between nodes directly

▷ ▷ **Syntax**

BD>**drawarc:** $\mathbf{B}_1$**,** $\mathbf{B}_2$**,** $\mathbf{I}_1$**,** $\mathbf{I}_2$ ↩

where B_1 *and* B_2 *are the names of* elements *or* bases*, and* I_1 *and* I_2 *are integers.*

◁ ◁

The DRAWARC: command is used to draw arcs explicitly between any two nodes B_1 and B_2 for an influence diagram. These nodes must already exist, but need not be drawn on the current influence diagram. The style and colour of the arc are determined by I_1 and I_2 respectively. I_2 is an integer representing a colour, as summarised in table 11.3. The possible values for I_1 are as follows.

$I_1 = 0$ gives a solid-line undirected arc.

$I_1 = 1$ gives a dotted-line undirected arc.

$I_1 = 2$ gives a solid-line directed arc, with an arrow at the end of the arc drawn from node B_1 to B_2.

$I_1 = 3$ gives a dotted-line directed arc, with an arrow at the end of the arc drawn from node B_1 to B_2.

11.6 Drawing arcs between nodes directly

▷ ▷ **Syntax**

BD>**line:** $\mathbf{R}_1$**,** $\mathbf{R}_2$**,** $\mathbf{R}_3$**,** $\mathbf{R}_4$**,** $\mathbf{I}_1$**,** $\mathbf{I}_2$ ↩

where $R_1, \ldots, R_4$ *are the equations which evaluate to real numbers such that* $0 \leq R_i \leq 1$*, and* I_1 *and* I_2 *are integers.*

◁ ◁

The LINE: command is used to draw lines explicitly between any two points for an influence diagram. A line is drawn between the points (R_1, R_2) and (R_3, R_4). The style and colour of the arc are determined by I_1 and I_2 respectively. I_2 is an integer representing a colour, as summarised in table 11.3. The possible values for I_1 are as follows.

$I_1 = 0$ gives a solid-line undirected arc.

$I_1 = 1$ gives a dotted-line undirected arc.

11.6.1 Removing nodes and arcs

▷ ▷ **Syntax**

BD>**xgrid: B1, B2,** ... ↩

where $B_1, B_2, \ldots$ *are the names of* elements *or* bases.

◁ ◁

We use the XGRID: command to remove node and arc definitions. All nodes associated with the named elements and bases, and any arcs connecting them to any other arcs, are removed. The XBASE: and XELEMENT: commands will also remove any arc and node definitions for the quantities to be deleted. Note that if a node has been associated with a base then the node remains defined whilst the base remains defined, whatever the contents of the base. See §6.2 for details of specifying collections for the definition part.

11.7 Producing canonical wheels

11.7.1 Overview

Instead of producing influence diagrams as we have described above, we may instead produce canonical wheel influence diagrams. These are the same as the influence diagrams described above (and share the properties of being influence diagrams in the classical sense) except that the shading of nodes is more detailed and has a different meaning. A simple switch, using the wheel control, determines whether the diagrams you produce have standard shading or canonical wheel shading. By default, all influence diagrams are of the standard variety. Other relevent controls are as follows. Node shading colours are controlled by the spokecolour control. The level of detail seen is controlled by the spokes control, and the visual impact of a given diagnostic is controlled by the spokeshade control. The setting of the overshade control has no effect when canonical wheels are being produced.

11.7.2 Graphic assessments of adjustments via canonical quantities

Consider a standard Bayes linear influence diagram for the adjustment of B by D, and then partially by F, and suppose also that observations d and f become available. The adjustment by D results in a portion of the B node's outer sector being shaded to show the proportion of uncertainty across the collection B, $\mathrm{RU}_D(B)$, resolved by adjusting by D. Additionally, the corresponding inner sector is shaded according to the magnitude and direction (too big/too small) of the size ratio for the adjustment, $\mathrm{Sr}_d(B)$. Both shadings (outer and inner) reflect summaries across the collection B assessed for D globally. A further partial adjustment by F induces similar shadings for the partial adjustment, so that an extra portion of B's outer node is shaded to reflect the extra variance resolution $\mathrm{RU}_{F/D}(B)$ from fitting on F in addition to D, and the corresponding inner node shading depicts the partial size ratio $\mathrm{Sr}_{f/d}(B)$.

The idea behind the canonical wheel influence diagrams is to partition the global summaries $\mathrm{RU}_D(B)$, $\mathrm{RU}_{F/D}(B)$, according to the canonical quantities generated by the two adjustments. This follows naturally as follows. Suppose that the adjustment of B has r canonical quantities $Z_1, \ldots, Z_r$ with resolutions $\lambda_1, \geq \ldots, \geq \lambda_r$. Then the global proportion of variance resolved, $\mathrm{RU}_D(B)$ is naturally partitioned into $\mathrm{RU}_D(B) = \sum \mathrm{RU}_D(Z_i) = \sum \lambda_i / r_B$, where $r_B = \mathrm{r}(\mathrm{Var}(B))$ is by convention the amount of prior uncertainty in the collection B. Therefore, the nodal area represented by $\mathrm{R}_D(B)$ can be partitioned into what we term r **spokes**, each of which is a sector corresponding to a canonical quantity Z_i, with outer node shading λ_i / r_B.

If there are observations then we also shade part of each inner spoke to show diagnostics. The diagnostics shown are essentially the values of the size ratios for the canonical quantities, and are thus more-or-less comparable with the diagnostic shadings shown for standard influence diagrams. A slightly different scaling strategy has been chosen, however, as follows. The proportion of area shaded depends on the value of the corresponding size ratio, $\mathrm{Sr}_d(Z_i)$, divided by a threshhold set by the spokeshade control, which is at 8.0 by default. Proportions larger than unity are clipped at unity. Therefore by default, size ratios of 4.0, 8.0, and 12.0 will result in 50%, 100%, and 100% inner sector shadings. Threshholds should be chosen so that a very surprising size ratio ("surprisingness" can vary from context to context) results in about full diagnostic shading, so that the degree of shading conveys the degree of alarm. Bear in mind that the prior expectation of every size ratio is unity. The colour of the shading can be altered using the spokecolour control.

The partial adjustment similarly results in a collection of partial canonical quantities which form a natural partition for the global summaries for the partial adjustment. Thus, we partition the sector previously corresponding to the partial resolved uncertainty into separate spokes, depending upon the number of partial canonical quantities, i.e. the rank of the partial resolution matrix. Size ratio diagnostics are shaded accordingly and similarly.

Sometimes the number of canonical quantities for an adjustment can be so large that too many spokes would be drawn on the canonical wheel, so offering little value as a means of understanding an adjustment. In such cases you can reduce the number of spokes that are drawn by using the spokes control. If you wish to see shadings for the first $q < r$ canonical directions only, use the spokes control to limit the number of spokes drawn to q canonical quantities. This will then produce node shadings for $q + 1$ spokes, with the extra shading in the final spoke aggregating the resolved uncertainty and diagnostic information for the last $q - r$ canonical directions. The amount of outer and inner shading drawn for the aggregated spokes will sum to the same amount of shading that would have been observed had all the spokes been drawn.

Note that the spokes are drawn anticlockwise in order of magnitude of the canonical resolutions. The areas given over to successive canonical quantities thus typically decline. Full diagnostic shading for a spoke (indicating very surprising changes in expectation) can have very little visual impact for canonical quantities with only small canonical resolutions: mostly we should not be too alarmed about bizarre occurrences in otherwise unimportant directions.

11.8 Defining titles for influence diagrams

▷ ▷ **Syntax**

1. `BD>`**ititle: T** ↩

2. `BD>`**ititle:** ↩

where T is any sequence of alphanumeric text.

◁ ◁

The ITITLE: command is used to define a title to be used for influence diagrams. As for the STRING:, PRINT:, TITLE: and PCTITLE: commands, items in square brackets, [...], are not parsed for substitutions, and the text does not have its blank spaces removed, or its capital letters changed to lower case.

The second form of the syntax, where the command is given without an argument, deletes any such title.

Chapter 12

Partial correlation and belief comparison diagrams

12.1 Partial correlation diagrams

12.1.1 Overview

Note that this command is under construction. Conditional dependencies between quantities and collections of quantities can be explored by making various transformations of their joint variance-covariance matrices. The transformations essentially reveal certain features of the underlying linear relationships between the variables, and these features can be represented on diagrams.

Suppose that a number of collections $B_1, B_2, \ldots$ together form a collection $B = E_1, E_2, \ldots, E_n$ of elements whose variance matrix is V. Firstly we transform to correlation form by determining $R = D^{-\frac{1}{2}} V D^{-\frac{1}{2}}$, where D is the $n \times n$ diagonal matrix whose entries are $D_{ii} = V_{ii}$. Next we determine the Moore-Penrose generalised inverse of R as follows. The pseudo-inverse obtained is $R^\dagger = Q \Lambda^\dagger Q^T$, where Λ is the diagonal matrix of ordered eigenvalues of R, and $\Lambda^\dagger$ has the non-zero diagonal elements inverted; and where Q is the matrix of corresponding orthonormal eigenvectors. Next we transform this generalised inverse itself into correlation form by calculating $\bar{R} = \bar{D}^{-\frac{1}{2}} R^\dagger \bar{D}^{-\frac{1}{2}}$, where $\bar{D}$ is the $n \times n$ diagonal matrix whose entries are $\bar{D}_{ii} = R^\dagger_{ii}$. However, we perform this final process only on the off-diagonal entries of $R^\dagger$, leaving alone the diagonal entries. Finally we multiply each off-diagonal entry by -1. The matrix G so constructed has the following properties.

1. the diagonal values G_{ii} are such that
$$G_{ii} = \frac{1}{1 - R^2}$$
where R is the multiple correlation coefficient between E_i and the remaining $n - 1$ elements.
2. the off-diagonal values are $G_{ij} = G_{ji}$, representing the partial correlation coefficient between E_i and E_j given the remaining $n - 2$ elements.

In [B/D] we make available G given $B_1, B_2, \ldots$ by using the PCDIAG: command in combination with the pcdest control, which sets the belief store in which these results are to be stored. The belief store containing the original variance matrix V can be switched using the pcsource control. The partial correlation diagram summarising the linear relationships is obtained by using the PCDIAG: command in combination with the pcdiag option. A title may be defined for this diagram by using the PCTITLE: command. Linear relationships (partial) are shown as arcs on the diagram. The number of arcs shown depends upon the value of the pcarc control: arcs corresponding to partial correlations less than this value are supressed. The diagram itself needs to be designed beforehand, using the GRID: or, more conveniently, the GRID0: command.

12.1.2 Producing the diagram

▷ ▷ **Syntax**

BD>**pcdiag:** B_1, B_2, ... ↩

where $B_1, B_2, \ldots$ *are the names of* bases *or* elements.

◁ ◁

The PCDIAG: command is used to calculate and store the partial correlation matrix corresponding to a variance-covariance matrix, and to display the calculations on a partial correlation diagram. $B_1, B_2, \ldots$ are assumed to be a list of elements or base names which jointly form the collection $B = E_1, E_2, \ldots$. A variance matrix is assumed to have been specified for this collection B, and to be stored currently in the belief store pointed to by the pcsource control (store 1 by default). The results are available either numerically or in the form of a diagram or both.

Firstly, if the pcdest control has a value equal to a belief store number, the numerical results detailed above as the matrix G are stored as beliefs in this belief store.

Secondly, if the pcdiag option is switched on, a partial correlation diagram is output, together with any title defined using the PCTITLE: command. The diagram consists of the following. Each of the collections $B_1, B_2, \ldots$ is represented as a node on the diagram. The nodes need also to have been given coordinates, colour, and so forth, using the GRID: or GRID0: commands. (Any arcs defined within the GRID: command are ignored.) The nodes are connected by arcs if the partial correlation between the two nodes given the rest exceeds the threshhold given by the pcarc control, whose value is zero by default.

The arcs are labelled by a small circle showing the degree of partial correlation. We shade the circle according to the absolute magnitude of the correlation, so that correlations of 1 and -1 result in a fully shaded circle. We shade differently, according to direction. For positive correlations, we shade a corresponding proportion of the circle anti-clockwise, starting from 0 degrees. For negative correlations, we shade a corresponding proportion of the circle clockwise, starting from 0 degrees. The size of the circle can be changed using the pcradius control.

12.2 Defining titles for partial correlation diagrams

▷ ▷ **Syntax**

1. `BD>`**pctitle: T** ↩

2. `BD>`**pctitle:** ↩

where T *is any sequence of alphanumeric text.*

◁ ◁

The PCTITLE: command is used to define a title to be used for partial correlation diagrams. As for the STRING:, ITITLE:, TITLE:, and PRINT: commands, items in square brackets, [...], are not parsed for substitutions, and the text does not have its blank spaces removed, or its capital letters changed to lower case.

The second form of the syntax, where the command is given without an argument, deletes any such title.

12.3 Belief comparison diagrams

12.3.1 Overview

Note that this command is under construction. The purpose of belief comparison diagrams is to summarise relationships between alternative variance specifications and prediction error for a collection of quantities. We have in mind that there are two alternative information sources, D_1 and D_2, with possibly different specifications about and between B and D_1 on the one hand, and about and between B and D_2 on the other. We will treat the two scenarios as model 1 and model 2 respectively.

Assume that the two alternative variance specifications given for collections B, D_1 and B, D_2 are respectively $\mathrm{Var}_1([B\ D_1]^T)$ and $\mathrm{Var}_2([B\ D_2]^T)$. Initially we will be concerned with $\mathrm{Var}_1(B) \neq \mathrm{Var}_2(B)$, but we allow the possibility $D_1 \equiv D_2$. We also restrict ourselves to the case where data is available on

D_1 and D_2, so that we can calculate both possible adjusted expectations for B, and that we also have available the observed value b of B.

Under model 1, suppose now that the collection B is adjusted by collection D_1. This gives rise to an adjusted version $B^{(1)} = B - \mathrm{E}_{D_1}(B)$ which is some vector of linear combinations of the elements in B. This quantity is observed to be $b^{(1)} = b - \mathrm{E}_{d_1}(B)$, the vector of prediction errors for model 1. We now calculate $V_{11} = \mathrm{Var}_1(B^{(1)})$ and $V_{12} = \mathrm{Var}_2(B^{(1)})$ to be its variance matrix under model 1 and model 2 respectively.

We now carry out a comparison of these two alternative variance specifications for the adjusted version of B. Let $\lambda^{(2)}_{min}$ and $\lambda^{(2)}_{max}$ be the smallest and largest canonical quantities (generalised eigenvalues) for this comparison, corresponding to canonical quantities (generalised eigenvectors) W_{min}, W_{max} whose observed magnitudes are w_{min} and w_{max} respectively. The eigenvectors are uncorrelated under both models. The eigenvalues $\lambda^{(2)}_i = \mathrm{Var}_2(W_i)$, whereas the W_i's are normed to have variance unity (or, pathologically, zero) under model 1: $\mathrm{Var}_1(W_i)$=1. If model 1 is correct, we should expect to see w_i^2 about equal to 1. Observing w_i^2 close to $\lambda^{(2)}_i$ suggests that the data give more support to model 2. Observing w_i^2 distant from both unity and $\lambda^{(2)}_i$ suggests that the data support neither model.

The upper semicircle of each node on the comparison diagram summarises these features. The top left quadrant is, from the centre outwards, shaded (white,red) in the proportions $(1, 1/\sqrt{\lambda^{(2)}_{max}})$. The observed value $|w_{max}|$ is plotted according to the current value of the cdscaler control, which treats the radius in the top left quadrant as being equal to (by default) three standard deviations relative to model 2. Values of w_{max} larger than three standard deviations are plotted on the quadrant boundary to indicate support for neither model. The top left quadrant is plotted only if $\lambda^{(2)}_{max} > 1$. A large amount of red indicates that model 2 has much more variance for some linear combinations of B than does model 1.

The top right quadrant is, from the centre outwards, shaded (white,blue) in the proportions $(\sqrt{\lambda^{(2)}_{min}}, 1)$. The observed value $|w_{min}|$ is plotted according to the current value of the cdscaler control, which treats the radius in this quadrant as being equal to (by default) three standard deviations relative to model 1 (i.e. 3, as the standard deviation is unity under model 1). Values of w_{min} larger than three are plotted on the quadrant boundary to indicate support for neither model. The quadrant is plotted only if $\lambda^{(2)}_{min} < 1$. A large amount of blue indicates that model 2 has much less variance for some linear combinations of B than does model 1.

We repeat the adjustment process under model 2, arriving at the vector $B^{(2)} = B - \mathrm{E}_{D_2}(B)$, the observed vector $b^{(2)} = b - \mathrm{E}_{d_2}(B)$, and variance matrices V_{21} and V_{22}.

We now carry out a comparison of these two alternative variance specifications for this second adjusted version of B. Let $\lambda^{(1)}_{min}$ and $\lambda^{(1)}_{max}$ be the smallest and largest canonical quantities (generalised eigenvalues) for this comparison, corresponding to canonical quantities (generalised eigenvectors) Z_{min}, Z_{max} whose observed magnitudes are z_{min} and z_{max} respectively. The eigenvectors are uncorrelated under both models. The eigenvalues $\lambda^{(1)}_i = \mathrm{Var}_1(W_i)$, whereas the Z_i's are normed to have variance unity (or, pathologically, zero) under model 2: $\mathrm{Var}_2(Z_i)$=1. If model 2 is correct, we should expect to see z_i^2 about equal to 1. Observing z_i^2 close to $\lambda^{(1)}_i$ suggests that the data give more support to model 1. Observing z_i^2 distant from both unity and $\lambda^{(1)}_i$ suggests that the data support neither model.

The lower semicircle is now drawn similarly to the upper semicircle. The bottom left quadrant is, from the centre outwards, shaded (white,blue) in the proportions $(1, 1/\sqrt{\lambda^{(1)}_{max}})$. The observed value $|z_{max}|$ is plotted according to the current value of the cdscaler control, which treats the radius in this quadrant as being equal to (by default) three standard deviations relative to model 1. Values of z_{max} larger than three standard deviations are plotted on the quadrant boundary to indicate support for neither model. This quadrant is plotted only if $\lambda^{(1)}_{max} > 1$. A large amount of blue indicates that model 1 has much more variance for some linear combinations of B than does model 2.

The bottom right quadrant is, from the centre outwards, shaded (white,red) in the proportions $(\sqrt{\lambda^{(1)}_{min}}, 1)$. The observed value $|z_{min}|$ is plotted according to the current value of the cdscaler control, which treats the radius in this quadrant as being equal to (by default) three standard deviations relative to model 2. Values of z_{min} larger than three are plotted on the quadrant boundary to indicate support for neither model. The quadrant is plotted only if $\lambda^{(1)}_{min} < 1$. A large amount of red indicates that model 1 has much less variance for some linear combinations of B than does model 2.

Lines are drawn to connect the observations in diagonally opposite quadrants to indicate that the quadrants convey similar kinds of information. Observations so large as to imply plotting beyond the boundary of a node are plotted on the boundary and shaded a different colour. The radius of the small circle representing the observation can be changed via the cdradius control.

12.3.2 Producing the diagram

▷ ▷ **Syntax**

1. BD>**cdiag:** B, I_1, R_1,R_2, R_3,R_4,R_5,R_6, R_7,R_8,R_9,R_{10}, R_{11}, R_{12} ↩

2. BD>**cdiag:** B, **1**, R_1,R_2, R_3,R_4,R_5,R_6 ↩

where B is the name of a base *or* element*, I_1 is an integer, and $R_1, \ldots, R_{12}$ are real numbers.*

◁ ◁

The CDIAG: command is used to construct the comparison diagram. Note that the GCLEAR: command can be used to clear the graphics screen. Furthermore, text can be written to the screen by using the GTEXT: command. Refer to the cdscaler and cdradius controls for changing the relative plotted positions of the observed eigenvectors and the size of the circles drawn to mark them.

All the features summarised on the diagram must be supplied directly: the command carries out no further adjustments of belief constructions. When the base B consists of a single element, a shorter version of the command is required. For both, B is the name of the collection being adjusted. The base B is represented by a node whose coordinates and so forth must have been defined beforehand by issuing an appropriate NODE: or GRID: or GRID0: command.

The number of elements in the base should be given as I_1. The remaining numbers supplied are the eigenvalues and observed eigenvectors as follows.

- For more than one element:

$$R_1 = \lambda^{(2)}_{min}, R_2 = (0 \text{ or } 1), R_3 = |w_{min}|$$

$$R_4 = \lambda^{(2)}_{max}, R_5 = (0 \text{ or } 1), R_6 = |w_{max}|$$

$$R_7 = \lambda^{(1)}_{min}, R_8 = (0 \text{ or } 1), R_9 = |z_{min}|$$

$$R_{10} = \lambda^{(1)}_{max}, R_{11} = (0 \text{ or } 1), R_{12} = |z_{max}|$$

- For one element there is only one eigenvalue for each comparison, and one observed eigenvector:

$$R_1 = \lambda^{(2)}, R_2 = (0 \text{ or } 1), R_3 = |w|$$

$$R_4 = \lambda^{(1)}, R_5 = (0 \text{ or } 1), R_6 = |z|$$

For each case above, by (0 or 1) we mean the normalised variance with respect to the denominator variance matrix. The required value will be unity except for pathological cases.

12.3.3 Example

For an example we adapt a problem considered in [49] concerning predicting the yields of a main product and a by-product at different temperature settings for a certain chemical process. The construction of the model is shown in Figure 12.1 and Figure 12.2. The model construction is identical to that of [49], with the exception that an alternative model is created simultaneously in belief store number 2, by using the FVAR: command to specify alternative variances for the error components in model 2.

For this simple example, we will compare adjustments for the four quantities in the base b by the two quantities in the base D, and then compare the adjustment of the element $b.3$ by D. Figure 12.3 shows how we arrange matters before using the general comparison macro shown in Figure 12.4. The name of the base being adjusted is supplied as the string *main*, whereas the name of the collection representing the information is supplied as the string *info*. At this point it is also necessary to ensure that node coordinates are given for the collection supplied as *main*. Finally, control passes to the *@compare* macro to perform the adjustments, carry out the comparisons, and draw the diagram. Notice that the macro is not completely general, as it is assumed therein that the alternative specifications are in belief stores 1 and 2.

Figure 12.1: Model construction, part I

```
element: <b> b.1=75, b.2=40, b.3=20, b.4=-30
var: v(1,b)
 4
-6   225
-1     0     1
 0   -90  -2.4  144
var: v(2,b)
 1
-3   100
-0.5   0     1
 0   -50     3  100

data b.1(1)=70 | data b.2(1)=45 | data b.3(1)=22 | data b.4(1)=-27

assign: y.i=b.1+(x(.i))b.2+e.i
assign: z.i=b.3+(x(.i))b.4+f.i

fvar: v(1,e.i,e.j)=(.i=.j)*6.25  |  fvar: v(2,e.i,e.j)=(.i=.j)*4
fvar: v(1,f.i,f.j)=(.i=.j)*4     |  fvar: v(2,f.i,f.j)=(.i=.j)*4
fvar: v(1,e.i,f.j)=(.i=.j)*2.5   |  fvar: v(2,e.i,f.j)=(.i=.j)*2
```

Figure 12.2: Model construction, part II

```
data: <x,ydata,zdata>12
  161.30   63.70   20.30
  164.00   59.50   24.20
  165.70   67.90   18.00
  170.10   68.80   20.50
  173.90   66.10   20.10
  176.20   70.40   17.50
  177.60   70.00   18.20
  181.70   73.70   15.40
  185.60   74.10   17.80
  189.00   79.60   13.30
  193.50   77.10   16.70
  195.70   82.80   14.80

for: i=1,1,12 | data: x([i])=(x([i])-177.86)/100
index: i=1,1,12
cobuild: y.i,z.i

for: i=1,1,12
   data: y.[i](1)=ydata([i])
   data: z.[i](1)=zdata([i])
end:

base: D=Z.1,Y.1
```

Figure 12.3: Arranging the comparison

```
string info=D
string main=b
node b,0.2,0.5,0.15,4,0.2,0.10
m @compare
string main=b.3
node b.3,0.8,0.5,0.15,4,0.8,0.10
m @compare
return
```

Figure 12.4: Making the comparison

```
@compare
base mainb={[main]}
c %n=countb(mainb)
keep e=EDG
keep bcd=w

string source=1
control priorvar=1,betweenvar=1,infovar=1
control compare=2,1
m @maincomp
string source=2
control priorvar=2,betweenvar=2,infovar=2
control compare=1,2
m @maincomp
if (%n>1)
   cdiag [main],%n,%ltopmin1,%lbotmin1,%minsum1,%ltopmax1,%lbotmax1,%maxsum1, &
                   %ltopmin2,%lbotmin2,%minsum2,%ltopmax2,%lbotmax2,%maxsum2
else
   cdiag [main],1,%ltopmin1,%lbotmin1,%minsum1,%ltopmin2,%lbotmin2,%minsum2
endif

return

@maincomp
adjust [mainb/[info]]
for i=1,1,%n | build K.[i]=[<mainb:[i]>]-!EDG[i]
xbase maink
for i=1,1,%n | base maink=maink,K.[i]
compare maink
c minsum[source]=0
c maxsum[source]=0
for i=1,1,%n | c minsum[source]=%minsum[source]+ascf(w1,K.[i])*K.[i](1)
for i=1,1,%n | c maxsum[source]=%maxsum[source]+ascf(w[(%n)],K.[i])*K.[i](1)
c minsum[source]=abs(%minsum[source])
c maxsum[source]=abs(%maxsum[source])
c ltopmin[source]=bcd1(1)
c ltopmax[source]=bcd1([(%n)])
c lbotmin[source]=bcd2(1)
c lbotmax[source]=bcd2([(%n)])
return
```

Chapter 13

Miscellaneous numerical commands

13.1 Checking the coherence of belief specifications

▷ ▷ **Syntax**

BD>**coherence: N1,** [**N2,** ...] ↩

where $N1$, $N2$, ... , *are the names of* elements *or* bases.

◁ ◁

The intention of this command is to perform coherence checking prior to carrying out an adjustment of a collection B by a collection D. To use the command, it is necessary first that the collection B be prepared for adjustment. That is, B must be currently defined to be the collection being adjusted. This is the case when you have already performed an adjustment, as in

BD>**adjust:** [**B/F**] ↩

Otherwise, a collection can be prepared for adjustment by issuing the command

BD>**adjust:** [**B/**] ↩

Suppose that we gather the elements and bases supplied as arguments to the command into the collection D. The COHERENCE: command now checks the coherence of the variance matrix specified over B and D jointly.

The effect of the command is essentially to perform the same kind of preliminary coherence check that is ordinarily made when you issue an ADJUST: type command. See §6.2 for details of specifying collections for the definition part.

The command will be applied typically to checking the coherence of beliefs expressed about second-order exchangeable sequences as outlined in §16.1. Suppose that $D_{(t)}$ is the t^{th} observation on the vector D. The variance matrices of interest are $G+U = \text{Var}(D_{(t)})$, $G = \text{Cov}(D_{(t)}, D_{(r)})$, and U, the difference between them. Usually, $U+G$ and G are specified directly, rather than U and G being specified.

To use the COHERENCE: command, you must set the infovar control to point to the belief store containing $U+G$ and the modelvar control to point to the belief store containing G. One coherence check then is to check that both G and U are nonnegative definite.

Note that, to use the infinite exchangeability representation, the collection formed by B and the *mean components* of D must have a nonnegative definite joint variance matrix.

To check that a single variance matrix is nonnegative definite, the COHERENCE: command can be used in exactly the same way, but with the modelvar and infovar controls pointing to the same belief store. In such a case the variance matrix for the collection D in the inidicated belief store has its coherence checked.

Data are not taken into account.

13.2 Eigendecomposition of a real symmetric matrix

▷ ▷ **Syntax**

1. BD>**eigen: v(I,B)** ↩

2. BD>**eigen: r(I,B)** ↩

where I is a valid belief store number and B is the name of a base *or* element.

◁ ◁

The EIGEN: command is used to obtain the eigenvectors and eigenvalues of a real symmetric matrix: typically the variance matrix specified over the collection B in the belief store I. (However, provided that the matrix is real and symmetric, the EIGEN: command can be applied to, and will give correct results for, for example, matrices having some eigenvalues less than zero, or all equal to zero.)

The second form of the syntax is used to perform the calculation of the eigenvalues and eigenvectors of the matrix in *correlation* form.

If a matrix is supplied with the property that all of its entries are zero, the eigenvalues are taken to be zero; and the matrix of corresponding eigenvectors is the full-rank identity matrix.

13.2.1 Viewing and retaining the eigenstructure

If you wish to see the eigenvalues and, optionally, the eigenvectors as determined by the command, you must use the `eig` or `eig+` option beforehand. Otherwise, some of the results from the EIGEN: command are available as [B/D] operators and operands. They are as follows:

- the rank of the matrix is given by **eigrank**
- the trace of the matrix is given by **eigtr**
- the eigenvalues are retained as **eig**(I), I=1 ... **eigrank**.

The Eigenvectors may be retained optionally as assignments by using the eig argument to the KEEP: command, before issuing the EIGEN: command.

As an example, suppose that we wish to calculate and display the eigenstructure of a variance matrix specified over some collection of n quantities $B = [B_1 \ldots B_n]$ in belief store 2; and that we wish to retain the eigenvectors for later use as the assignments $Z1, Z2, \ldots, Zn$. The following sequence of commands achieves this aim:

BD>**keep: eig = Z** ↩
BD>**option: eig+** ↩
BD>**eigen: v(2,B)** ↩

13.3 Calculating Moore-Penrose generalised inverses

▷ ▷ **Syntax**

1. BD>**invert: v(I1,B1)** ↩

2. BD>**invert: v(I1,B1) , v(I2,B2)** ↩

3. BD>**invert: r(I1,B1)** ↩

4. BD>**invert: r(I1,B1) , v(I2,B2)** ↩

where $I1$ and $I2$ are valid belief store numbers and $B1$ and $B2$ are the names of bases *or* elements.

◁ ◁

The INVERT: command is used to calculate the Moore-Penrose generalised inverse of a real symmetric matrix. For nonsingular matrices, this yields the usual matrix inverse. (For nonsingular matrices, the algorithm employed may be somewhat inefficient.)

In the first form of the syntax, the variance matrix specified over $B1$ in belief store $I1$ is taken; its generalised inverse calculated; and then this generalised inverse overwrites the original. In the second form of the syntax, the calculated generalised inverse is stored instead as though it were a variance matrix specified over the collection $B2$. In this case the two collections $B1$ and $B2$ need not be disjoint, but they do need to be of the same dimension.

The third and fourth forms of the syntax repeat the first and second respectively, except that the original matrix is transformed into correlation form before the generalised inverse is calculated. The generalised inverse is then stored as required.

The INVERT: command is unaffected by the LOCK: command.

Supposing that the matrix to be inverted is V of dimension n, the pseudo-inverse obtained is $V^\dagger = R\Lambda^\dagger R^T$, where Λ is the diagonal matrix of ordered eigenvalues of V, and $\Lambda^\dagger$ has the non-zero diagonal elements inverted; and where R is the matrix of corresponding orthonormal eigenvectors. The generalised inverse of the null matrix we take to be the null matrix.

13.4 Multiplying beliefs together

▷ ▷ **Syntax**

1. BD>**matmult: v(I1,B1,B2)=v(I2,B3,B4)*v(I3,B5,B6)** ↩

2. BD>**matmult: v(I1,B1,B2)=v(I2,B3,B4)*r(I3,B5,B6)** ↩

3. BD>**matmult: v(I1,B1,B2)=r(I2,B3,B4)*v(I3,B5,B6)** ↩

4. BD>**matmult: v(I1,B1,B2)=r(I2,B3,B4)*r(I3,B5,B6)** ↩

where $I1, I2, I3$ are valid belief store numbers, and $B1, \ldots, B6$ are the names of bases *or* elements. *In all parts, $v(1, B1, B1)$ may be abbreviated to $v(1, B1)$.*

◁ ◁

The MATMULT: command is used to multiply beliefs together. The first form of the syntax essentially defines

$$\mathrm{Cov}(B1, B2) = \mathrm{Cov}(B3, B4) \times \mathrm{Cov}(B5, B6),$$

where $B1, \ldots, B6$ represdent collections, although the numbers involved may not actually be covariances per se. The left hand side of the equality defines the place where the result of the multiplication is to be stored, and the right hand side defines the product to be stored.

The dimensions of the collection must be compatible for the multiplication. Thus, the dimensions of $B1$ and $B3$ must match; the dimensions of $B2$ and $B6$ must match; and the dimensions of $B4$ and $B5$ must match.

The remaining forms of the syntax show how we ask for transformation to correlation form before carrying out the multiplication.

An error will occur when a part of the result of the product is asymmetric, and the intended storage position requires symmetric beliefs. An illustration of the kind of problem that might arise is given in §6.8.5.

Chapter 14

General commands

14.1 Saving and restoring sessions

It is not possible to save a [B/D] session and then to restore it fully. However, there are limited SAVE: and RESTORE: commands which save and restore the beliefs and data created, but generally not the results of any adjustments. These commands will be extended in due course.

14.1.1 Restore beliefs and data

▹ ▹ **Syntax**

BD>**restore: NNNN** ↩

where NNNN is a sequence of alphanumeric characters such that the names NNNN.r *and* NNNN.c *are legal filenames for the platform on which [B/D] is being run.*

◃ ◃

The RESTORE: command is used to restore data and belief specifications previously saved using the SAVE: command. If the corresponding SAVE: command was as in the example

BD>**save: newfile** ↩

then the RESTORE: command must be

BD>**restore: newfile** ↩

Note in particular that the restored specifications **overwrite** any previously existing specifications with the same names, but leave alone any other previously existing specifications.

14.1.2 Save beliefs and data

▹ ▹ **Syntax**

BD>**save: NNNN** ↩

where NNNN is a sequence of alphanumeric characters such that the names NNNN.r *and* NNNN.c *are legal filenames for the platform on which [B/D] is being run.*

◃ ◃

The SAVE: command creates two files in the current directory. There must be sufficient filestore to save the two files, whose sizes will depend on the number of belief and data specifications to be saved. The command saves the following quantities:

- expectations in all expectation stores;
- variances and covariances in all expectation stores;
- data, including factors;

• the base structure.

Note in particular that the command does not currently save any information from adjustments or nodal information.

14.2 Repeat last command

▷ ▷ **Syntax**

BD>**+:** [**Arguments**] ↩

where Arguments are arguments particular to the command being repeated.

◁ ◁

The +: command represents shorthand for the last command issued. For example, the sequence of commands

BD>**print: This is an example:** ↩
BD>**+: and so is this!** ↩
BD>**c: beethoven=9** ↩
BD>**+: mozart=41** ↩

has the same effect as the sequence of commands

BD>**print: This is an example:** ↩
BD>**print: and so is this!** ↩
BD>**c: beethoven=9** ↩
BD>**c: mozart=41** ↩

For macro development this feature is best avoided for it can lead to confusion with the arithmetic operator, and is not amenable to searching for a particular command. Note that in the command sequence

BD>**if: %fred=3 — print: hello** ↩
BD>**+: goodbye** ↩

the repetition refers to the immediately preceding PRINT: command only.

14.3 Partially restarting the program

▷ ▷ **Syntax**

BD>**clear:** ↩

◁ ◁

The CLEAR: command, which has no arguments, is used to restart the program afresh, with some exceptions. Firstly, any external files introduced when starting the program, or later via the CHANNEL: command, are left alone. Consequently, any macros and so forth remain available after clearing.

If the program was started initially with a log-in macro on a file on channel 1, this macro is not re-run. The history buffer is flushed to a history file if one has been defined, and the buffer is then emptied, and the storage of screen commands is switched off. The output channel is not reset to the screen.

The CLEAR: command does not affect the current settings of the commands affecting program flow. That is, the following constructs remain unaffected when the CLEAR: command is issued.

- IF: statements;
- FOR: statements;
- REPEAT: statements;
- WHILE: statements;

- the current output channel;
- macro-switching

Thus, it is possible to issue the CLEAR: command within a macro, for example within an active IF: statement. The RESTART: command is similar, but also clears currently-defined loops and the handling of macros.

14.4 Exporting data and beliefs

The EXPORT: command is used to output certain quantities to files. It is possible to output data, the names of elements together with their expectations, covariance matrices, and correlation matrices. Various switches (d,n,e,t,r,v) are used to select the type of export. We deal with each of these separately below. There are some features common to each type of export. Firstly, any numbers exported are output with a number of decimal places set by the dplaces control. Secondly, all items exported are output via one of the output channels. Thus, prior to output, two steps must be followed before anything can be exported:

1. an output channel must be associated with an external file using the CHANNEL: command;
2. you must tell the program which output channel you wish to use for exported quantities, using the ECHANNEL: command.

As an example, suppose that we wish to export items to an external file called `exported` in the current directory, and suppose also that we use output channel 3 for this purpose. Then we could issue the following commands:

BD>**channel: o3=exported** ↩
BD>**echannel: (3)** ↩

We treat the channel for exported quantities pretty much as we treat ordinary output channels. Thus, exported quantities are typically appended to whatever has been written already by the program to the output channel. In this way we can generate on an external file a sequence of exported quantities from successive EXPORT: commands. For this reason, it is possible to define a label which will be exported as the first line of output.

14.4.1 Exporting data

▷ ▷ **Syntax**

1. BD>**export: L (d) D1, D2,** ... ↩

2. BD>**export: (d) D1, D2,** ... ↩

3. BD>**export: L (d)** ↩

4. BD>**export: (d)** ↩

where d is a switch; $D1, D2, \ldots$ *, are the names of* data–carriers *or* bases*; and L is the name of a* file label.

◁ ◁

The first form of the command is used to export data so that they follow a label. Typically the label will be a string of alphabetic characters preceded by the symbol '@' (this symbol is used as the prefix recognised by [B/D] as implying that the remainder of a string is a label name). The label is output as one line of text. The second form of the command is identical to the first except that no label is defined, and no label will be exported.

On successive lines the data are exported, one line per case. For example, if the names of two data-carriers $D1$ and $D2$ are given, and if each of these has ten observations defined ($D1_1, \ldots, D1_{10}$ and $D2_1, \ldots, D2_{10}$) then the output will consist of ten rows of two columns, with the first column corresponding to $D1$ and so forth.

Where $D1$, etc., are the names of bases, all the elements possessing data in the named base are marked for their data to be output. The latter two forms of the command do not specify the particular names of data to be exported, and are used as shorthand to mean that *all* data-carriers should be exported. See §6.2 for details of specifying collections for the definition part.

The data-carriers are output according to the ordering convention of data name, and their ordering in the EXPORT: command line is irrelevant.

14.4.2 Exporting element names

▷ ▷ **Syntax**

1. `BD>`**export: L (n) N1, N2, …** ↩

2. `BD>`**export: (n) N1, N2, …** ↩

where n is a switch; $N1, N2, \ldots$, are the names of elements *or* bases*; and L is the name of a* file label*. Base notation (see §6.2.4) can be used with the names.*

◁ ◁

The first form of the command is used to export names so that they follow a label. Typically the label will be a string of alphabetic characters preceded by the symbol '@' (this symbol is used as the prefix recognised by [B/D] as implying that the remainder of a string is a label name). The label is output as one line of text. The second form of the command is identical to the first except that no label is defined, and no label will be exported.

The list of element names and bases, $N1$, $N2$, … , defines the collection of elements to be exported. It is an error if the collection so defined is empty. The elements in the collection are exported in the order according to the ordering convention (so their ordering in the EXPORT: command line is irrelevant). Each name is output on a separate line.

14.4.3 Exporting element expectations

▷ ▷ **Syntax**

1. `BD>`**export: L (e[S]) N1, N2, …** ↩

2. `BD>`**export: (e[S]) N1, N2, …** ↩

where e is a switch; $N1, N2, \ldots$, are the names of elements *or* bases*; the optional argument S is an expectation store number; and L is the name of a* file label*. Base notation (see §6.2.4) can be used with the names.*

◁ ◁

The first form of the command is used to export element expectations (not preceded by the element names) so that they follow a label. Typically the label will be a string of alphabetic characters preceded by the symbol '@' (this symbol is used as the prefix recognised by [B/D] as implying that the remainder of a string is a label name). The label is output as one line of text. The second form of the command is identical to the first except that no label is defined, and no label will be exported.

The list of element names and bases, $N1$, $N2$, … , defines the collection of elements to be exported. It is an error if the collection so defined is empty. The expectations for the elements in the collection are exported in the order according to the ordering convention (so their ordering in the EXPORT: command line is irrelevant). Each expectation is output on a separate line. If no optional argument S is given in the command, the expectations are taken from the default expectation store. Otherwise, expectations are taken from expectation store S.

14.4.4 Exporting element names and expectations

▹ ▹ **Syntax**

1. BD>**export: L (t[S]) N1, N2, . . .** ↩

2. BD>**export: (t[S]) N1, N2, . . .** ↩

where t is a switch; $N1, N2, \ldots$ *, are the names of* elements *or* bases*; the optional argument S is an expectation store number; and L is the name of a* file label*. Base notation (see §6.2.4) can be used with the names.*

◃ ◃

The first form of the command is used to export names and expectations so that they follow a label. Typically the label will be a string of alphabetic characters preceded by the symbol '@' (this symbol is used as the prefix recognised by [B/D] as implying that the remainder of a string is a label name). The label is output as one line of text. The second form of the command is identical to the first except that no label is defined, and no label will be exported.

The list of element names and bases, $N1$, $N2$, . . . , defines the collection of elements to be exported. It is an error if the collection so defined is empty. The elements in the collection are exported in the order according to the ordering convention (so their ordering in the EXPORT: command line is irrelevant). Each name is output on a separate line, together with its expectation. If no optional argument S is given in the command, the expectations are taken from the default expectation store. Otherwise, expectations are taken from expectation store S.

14.4.5 Exporting beliefs

▹ ▹ **Syntax**

1. BD>**export: L (vi) N1, N2, . . .** ↩

2. BD>**export: (vi) N1, N2, . . .** ↩

3. BD>**export: L (ri) N1, N2, . . .** ↩

4. BD>**export: (ri) N1, N2, . . .** ↩

where r and v are switches; $N1, N2, \ldots$ *, are the names of* elements *or* bases*; and L is the name of a* file label*. Base notation (see §6.2.4) can be used with the names. i is an integer representing a valid belief store number.*

◃ ◃

The first form of the command is used to export beliefs so that they follow a label. Typically the label will be a string of alphabetic characters preceded by the symbol '@' (this symbol is used as the prefix recognised by [B/D] as implying that the remainder of a string is a label name). The label is output as one line of text. The second form of the command is identical to the first except that no label is defined, and no label will be exported.

The list of element names and bases, $N1$, $N2$, . . . , defines the collection of elements for which a variance covariance matrix will be exported. It is an error if the collection so defined is empty. The elements in the collection are output in an the order given by the ordering convention (so that their ordering in the EXPORT: command line is irrelevant). The variance–covariance matrix for the collection corresponding to belief store number i is output, one number per line. For example, suppose that the collection being output is $X_1, \ldots, X_N$. On successive lines the output consists of the following variances and covariances: $\mathrm{Var}_i(X_1)$, $\mathrm{Cov}_i(X_1, X_2)$, . . . , $\mathrm{Cov}_i(X_1, X_N)$, $\mathrm{Cov}_i(X_2, X_1)$, $\mathrm{Var}_i(X_2)$, . . . , and so forth.

The latter two forms of the command, with r replacing v, duplicate the former two forms of the command, except that the corresponding beliefs are output as correlations rather than variances and covariances.

14.5 Pausing during output

▹ ▹ **Syntax**

BD>**pause:** ↩

◃ ◃

The PAUSE: command has no arguments. It is intended for use in macros when it is desirable to to split interactive screen output into chunks. When the command is encountered, the program prints a message on the screen and waits until the <RETURN> or <ENTER> key is pressed.

14.6 Displaying other output

▹ ▹ **Syntax**

BD>**print: This is some text, followed by the value of this constant: %fred is (%fred)** ↩
BD>**print: This will not throw a carriage-return or line-feed $** ↩

◃ ◃

The PRINT: command is used to output information to the screen: headings, titles, general information, and various numerical values. It has a special status in [B/D] as the argument to the print: command is not parsed in the same way as most of the other [B/D] commands. In fact parsing is limited to the following exceptions.

- Regarding the substitution of quantities in square brackets: the program first checks active string and for lists, and makes any substitutions for strings and control variables as necessary. Otherwise the quantity is printed as is.
- Material found inside round brackets is assumed to represent a valid [B/D] equation whose value is to be printed. Thus, the syntax

  BD>**print: text1 (E) text2** ↩

  is used to evaluate the value of the equation E, and to print this value after $text1$ and before $text2$. Prior to the determination of the equation, the text between the round brackets is parsed in the usual way to remove redundant spaces, turn text to lower case, and make any necessary substitutions for strings and control variables. The value is output with a minimum width set by the fwidth control, and to the number of decimal places specified by the dplaces control.
- You may wish to include the round brackets symbols as ordinary text. This is in any case possible for the closing round bracket symbol. For the opening round bracket symbol, this is achieved by prefixing it by the " symbol. Any character succeeding a " symbol is printed literally. (So to output the " symbol, you must use two side by side.) Examples are:

  BD>**print: (3+2) is an equation "(but this is not),** ↩

  which should produce the text ``5.0000 is an equation (but this is not)'',
  and

  BD>**print: ""See how we produce quotation marks"".** ↩

Otherwise, redundant spaces are *not* removed from the text in the PRINT: command, and the case of the text is not changed. This allows us to format text output nicely, capitalising wherever we want, and so forth.

The second form of the syntax, where the text following the command is terminated by the '$' symbol, supresses the carriage return and linefeed usually output. This allows text from different PRINT: statements to be output on the same line, as in the following example:

BD>**print: Beware the $** ↩
BD>**if: %x | print: dog! | else: | print: ides of March!** ↩

where $\%x$ might be some constant taking values zero or one.

14.7 Fully restarting the program

▷ ▷ **Syntax**

BD>**restart:** ↩

◁ ◁

The <u>RESTART:</u> command, which has no arguments, is used to restart the program afresh, with the particular exception that any external files introduced when starting the program, or later via the <u>CHANNEL:</u> command, are left alone. Consequently, any macros and so forth remain available after the restart.

If the program was started initially with a log-in macro on a file on channel 1, this macro is not re-run. The history buffer is flushed to a history file if one has been defined, and the buffer is then emptied, and the storage of screen commands is switched off. The output channel is reset to the screen.

The <u>CLEAR:</u> command is similar, but does not interfere with either currently-defined loops or the handling of macros.

14.8 Initialising random number generation

▷ ▷ **Syntax**

BD>**seed: I** ↩

where $0 \leq I < 131071$ *is a positive integer.*

◁ ◁

[B/D] contains a crude mechanism for generating uniformly or Normally distributed psuedo-random numbers. Uniform (0,1) pseudo-random numbers are generated by the linear congruential method (see, for example, [40]) via the sequence

$$\begin{aligned} x_{i+1} &= 257x_i + 65537 \mathsf{\ mod\ } 131071 \\ u_{i+1} &= x_{i+1}/131071. \end{aligned}$$

where x_{i+1} and x_i are the current and former values of the *seed*, and u_{i+1} is the generated random number. The <u>SEED:</u> command can be used to set the value of x_0 used to initiate the sequence of seed values.

Approximately Normal (0,1) psuedo-random numbers z_j are generated by the Polar method from a pair of Uniform psuedo-random numbers as follows:

$$z_j = \mathsf{cos}(2\pi u_2)\sqrt{-2\mathsf{ln}(u_1)}.$$

These uniform and Normal psuedo-random numbers are available in equations as the [B/D] operands <u>**urand**</u> and <u>**nrand**</u> respectively.

14.9 Terminating the program

▷ ▷ **Syntax**

BD>**stop:** ↩

◁ ◁

The <u>STOP:</u> command is used to terminate the program. When this command is issued, all variables are cleared internally and all files are updated if necessary and then closed.

Chapter 15

Examining inputs to the program

15.1 Introduction

▷ ▷ **Syntax**

1. BD>**look:** ↩

2. BD>**look: (argument** [**, argument,** . . .]**)** [**N1, N2,** . . .] ↩

where a number of arguments (each discussed below) appear in parenthesis, separated by commas, and $N1$, $N2$, . . . are the names of elements, bases, assignments, *and* data-carriers*, separated by commas.*

◁ ◁

The LOOK: command is used to examine various of the inputs to the program, together with other general information. The first form of the syntax simply lists the possible arguments available for general use of the command. The second form of the syntax is used to give a brief summary of various belief, data, functional, and other specifications and program features. Each argument represents a different set of summaries, whilst the base, element, and data-carrier names which follow the parenthesis limit certain summaries to those subsets specified. For the purposes of describing this command, we will refer to this subset as the *restriction*. Wildcard notation (§6.2.1) may be used in describing the restriction. Unrecognised arguments and options are ignored. The possible summaries are given by the arguments described as follows.

15.2 Possible arguments

All specifications

▷ ▷ **Syntax**

BD>**look: ($)** ↩

◁ ◁

The **$** argument is used as shorthand for all possible arguments to the LOOK: command. Any defined *restriction* will apply as appropriate to each successive argument.

Assignments

▷ ▷ **Syntax**

BD>**look: (a)** [**N1, N2,** . . .] ↩

where $N1$, $N2$, . . . are the names of assignments*, separated by commas.*

◁ ◁

To display *assigments* we use the a argument. This can be used with or without a *restriction*. Firstly, when this argument is used without a restriction, all of the assignments known to the program are displayed. Secondly, when a restriction is applied, the display is limited to the named assignments.

Where the assignments are known linear combinations of elements, these are displayed as such. Otherwise (for example, if parts of the assignment are functional) various information about the functionally defined assignment is displayed.

For functionally defined assignments, any name supplied in the restriction need contain only the stem of the assignment; any extraneous features are removed. For example, suppose that the functional assignment $x.i.j$ has been defined. To examine it, any one of the following 0commands would suffice:

1. `BD>`**look: (a)x** ↩

2. `BD>`**look: (a)x.3.4** ↩

3. `BD>`**look: (a)x.i.j** ↩

Bases

▹ ▹ **Syntax**

`BD>`**look: (b)** [**N1, N2,** ...] ↩

where $N1$, $N2$, ... *are the names of* bases*, separated by commas.*

◃ ◃

To display *bases* we use the b argument. This can be used with or without a *restriction*. Firstly, when this argument is used without a restriction, all of the assignments known to the program are displayed. Secondly, when a restriction is applied, the display is limited to the named bases. The names of the elements contained in each listed base are displayed in the order given by the ordering convention. See §6.2 for details of specifying collections for the definition part.

Influence diagram coordinates

▹ ▹ **Syntax**

`BD>`**look: (grid)** [**N1, N2,** ...] ↩

where $N1$, $N2$, ... *are the names of* bases *or* elements*, separated by commas.*

◃ ◃

The grid argument displays information set using the GRID: command. The argument can be used with or without a *restriction*. Firstly, when this argument is used without a restriction, all of the grid information known to the program is displayed. Secondly, when a restriction is applied, the information is limited to the named bases and elements. The information displayed is as follows.

1. the name of the base or element to be drawn as a node;
2. the radius of the node representing the base, where zero shows that the default radius to be used;
3. an integer representing the colour of the node as summarised in Table 11.3;
4. the (x, y) coordinate for the centre of the node;
5. a list of other elements and nodes which are to be connected with arcs drawn from this node, together with an integer representing the arc style as summarised in Table 11.4.

Constants

▹ ▹ **Syntax**

BD>**look: (c)** ↩

◃ ◃

To display *constants* we use the c argument. This argument is used without a *restriction*. The names of the constants are displayed in in the order given by the ordering convention, together with their values.

Commands

▹ ▹ **Syntax**

BD>**look: (commands)** ↩

◃ ◃

The commands argument is used to display a list of all the possible [B/D] commands, together with short descriptions. This argument is used without a *restriction*.

Controls

▹ ▹ **Syntax**

BD>**look: (controls)** ↩

◃ ◃

The various [B/D] controls may be listed by using the controls argument. This argument is used without a *restriction*. The names of the controls are displayed, together with a brief decsription. The nc argument displays the values of certain controls.

Data

▹ ▹ **Syntax**

BD>**look: (d)** [**N1, N2,** ...] ↩

where $N1$, $N2$, ... *are the names of* elements *or* bases *or* data-carriers*, separated by commas.*

◃ ◃

To display *data* we use the d argument. This can be used with or without a *restriction*. Firstly, when this argument is used without a restriction, all of the data known to the program are displayed. Secondly, when a restriction is applied, the display is limited to the data on the named data carriers. The names of the data carriers are displayed in the order given by the ordering convention. See §6.2 for details of specifying collections for the definition part.

The quantity of data output is controlled by the autoselect control as follows. If the autoselect control is on then all possible data is shown. If the control is off, data selection is as determined beforehand, according to prior usages of the SELECT: command.

Notice that the SUMMARY: command can be used to summarise various features of the data: means, standard deviations, correlations, and so forth.

Data stack checker

▹ ▹ **Syntax**

BD>**look: (dfstk)** ↩

◃ ◃

This is a programming tool enabling inspection of the starting positions and lengths of freed locations in the data storage area.

Data stack checker

▷ ▷ **Syntax**

BD>**look: (dstk)** ↩

◁ ◁

This is another programming tool enabling inspection of the starting positions and lengths of freed locations in the data storage area.

Element names, expectations, and standard deviations

▷ ▷ **Syntax**

1. BD>**look: (e)** [**N1, N2,** ...] ↩

2. BD>**look: (e[i])** [**N1, N2,** ...] ↩

where the option i is a valid expectation store number, and $N1$, $N2$, ... are the names of elements *or* bases*, separated by commas.*

◁ ◁

To display the names of currently defined *elements* we use the e argument. This can be used with or without a *restriction.* Firstly, when this argument is used without a restriction, all of the elements known to the program are displayed. Secondly, when a restriction is applied, the display is limited to the elements given by the collection represented in the *restriction.* The element names are displayed in the order given by the ordering convention. See §6.2 for details of specifying collections for the definition part.

The first form of the syntax displays element names together with their expectations in all expectation stores. The second form of the syntax displays element names, their expectations in expectation store i , and their standard deviations in belief store i.

Functions

▷ ▷ **Syntax**

BD>**look: (f)** ↩

◁ ◁

The f argument to the LOOK: command produces a list of *functions* defined so far. These are produced in the order given by the ordering convention. This argument is used without a *restriction.*

Functionally defined expectations

▷ ▷ **Syntax**

BD>**look: (fe)** ↩

◁ ◁

The fe argument to the LOOK: command produces a list of functionally-defined expectations. These are displayed in the order given by the ordering convention. This argument is used without a *restriction.*

External file attachments

▹ ▹ **Syntax**

BD>**look: (files)** ↩

◃ ◃

The files argument is used to display information about the link between [B/D]'s internal input and output channels and various external files. For each defined input and output channel, the full name of the associated external file is printed. Finally, the names of the external files associated with the log file (set using the log argument to the KEEP: command), the history file (set using the history argument to the KEEP: command), and the help file are displayed. This argument is used without a *restriction*.

For statement control variables

▹ ▹ **Syntax**

BD>**look: (for)** ↩

◃ ◃

Active [B/D] for-statements, together with the current values of the control variables and their initialising values, may be listed by using the for argument. This argument is used without a *restriction*.

Functionally defined beliefs

▹ ▹ **Syntax**

1. BD>**look: (fvar)** ↩
2. BD>**look: (fvar[i])** ↩

where the option i is a valid belief store number.

◃ ◃

The fvar argument is used to summarise the current functionally-defined variance–covariance specifications. The first form of the syntax displays functionally defined variances and covariances for all of the belief stores. The second form of the syntax limits the display to variances and covariances in belief store i. This argument is used without a *restriction*.

Index control variables

▹ ▹ **Syntax**

BD>**look: (i)** ↩

◃ ◃

Declared indices, as defined by INDEX: commands, together with the name of the control variables and their initialising values, may be listed by using the i argument. This argument is used without a *restriction*.

Obtaining user-defined retentions

▹ ▹ **Syntax**

BD>**look: (keep)** ↩

◃ ◃

The KEEP: command is used to select material to be retained in user-defined quantities. The keep argument displays a list of the possible retentions to the KEEP: command, with brief descriptions. This argument is used without a *restriction*.

Names of retained quantities

▹ ▹ **Syntax**

BD>**look: (kept)** ↩

◃ ◃

Several quantities may be stored under user-defined names via the KEEP: command. The kept argument here displays these user-defined names. This argument is used without a *restriction*.

Labels with comments

▹ ▹ **Syntax**

1. BD>**look: (l)** ↩

2. BD>**look: (l[i])** ↩

where the option i is a valid input channel number.

◃ ◃

The l argument is used to display labels, and is issued without a *restriction*. The first form of the syntax displays the labels currently defined on all open input channels. The second form of the syntax is used to display only the labels on input channel i.

Only those labels that are accompanied by comments (text that follows the label on the same line, and separated from the label by at least one space), are printed. The labels are produced in the order given by the ordering convention. This argument is used without a *restriction*.

The lall argument is similar, but displays all labels, regardless of any accompanying comments.

All labels

▹ ▹ **Syntax**

1. BD>**look: (lall)** ↩

2. BD>**look: (lall[i])** ↩

where the option i is a valid input channel number.

◃ ◃

The lall argument is used to display labels, and is issued without a *restriction*. The first form of the syntax displays the labels currently defined on all open input channels. The second form of the syntax is used to display only the labels on input channel i. This argument is used without a *restriction*.

All labels are printed, in the order given by the ordering convention. The l argument is similar, but displays only those labels accompanied by comments.

Program restrictions

▹ ▹ **Syntax**

BD>**look: (program)** ↩

◃ ◃

To display the most important run-time restrictions we use the program argument. This argument is used without a *restriction*. The information displayed concerns the following.

- the total number of data locations available, and whether these are stored on disk or in memory;

- the total number of belief stores available, and whether these are stored on disk or in memory;
- the maximum number of buildable elements;
- the maximum number of elements that can form a base to be used in an adjustment;
- the maximum depth of iteration for iterative adjustment within the ITADJUST: command;
- the maximum length in characters of [B/D] names for elements, bases, and so forth;
- the maximum length in characters of the title to be used in a PLOT: command;
- the resolution of the low-resolution plotting grid;
- the total number of records retained in the screen history buffer.

Control values

▹ ▹ **Syntax**

BD>**look: (nc)** ↩

◃ ◃

The nc argument, which is used without a *restriction*, displays a list of the current values of the following controls:

- belief store sources for the variance specifications for general adjustments;
- belief store sources for the variance specifications for belief comparisons;
- The current default expectation store and default variance store.
- current number formatting for many of the numbers output by [B/D];
- the missing value marker;
- the default sample size for adjustments where no data are available.

Operands

▹ ▹ **Syntax**

BD>**look: (operands)** ↩

◃ ◃

The operands argument displays a list of [B/D] operands (for example **rmtr** and **prmrank**) usable in [B/D] *equations*, together with brief descriptions of them. This argument is used without a *restriction*.

Operators

▹ ▹ **Syntax**

BD>**look: (operators)** ↩

◃ ◃

The operators argument displays a list of [B/D] operators (for example **eig** and **centile**) usable in [B/D] *equations*, together with brief descriptions of them. This argument is used without a *restriction*.

Options

▹ ▹ **Syntax**

BD>**look: (options)** ↩

◃ ◃

The options argument displays a list of the possible arguments to the OPTION: command, with brief descriptions. This argument is used without a *restriction*.

External file paths

▷ ▷ **Syntax**

BD>**look: (path)** ↩

◁ ◁

The path argument is used to display the current values of the search path variables BDINPUTS and BDMACROS. This argument is used without a *restriction*.

Correlations

▷ ▷ **Syntax**

1. BD>**look: (r)** [**N1, N2,** ...] ↩
2. BD>**look: (r[i])** [**N1, N2,** ...] ↩

where the option i is a valid belief store number, and $N1$, $N2$, ... are the names of elements *or* bases*, separated by commas.*

◁ ◁

To display beliefs (in correlation form) specified about currently defined *elements*, we use the r argument. This can be used with or without a *restriction*. Firstly, when this argument is used without a restriction, beliefs about all of the elements known to the program are displayed. Secondly, when a restriction is applied, the display is limited to beliefs about the elements given by the collection represented in the *restriction*. Beliefs are displayed for element names in the order given by the ordering convention. See §6.2 for details of specifying collections for the definition part.

The first form of the syntax displays correlations for all of the belief stores. The second form of the syntax limits the display to correlations in belief store i.

In the case of any "variance" terms being not positive, all the corresponding "correlation" terms are marked with a dash rather than a number.

Strings

▷ ▷ **Syntax**

BD>**look: (s)** ↩

◁ ◁

The s argument displays the names of all currently-defined strings, together with the title to be used for plots, influence diagrams, and partial correlation diagrams, if such titles has been defined using the TITLE:, ITITLE:, and PCTITLE: commands. This argument is used without a *restriction*.

Variances and covariances

▷ ▷ **Syntax**

1. BD>**look: (v)** [**N1, N2,** ...] ↩
2. BD>**look: (v[i])** [**N1, N2,** ...] ↩

where the option i is a valid belief store number, and $N1$, $N2$, ... are the names of elements *or* bases*, separated by commas.*

◁ ◁

To display beliefs (in variance-covariance form) specified about currently defined *elements*, we use the v argument. This can be used with or without a *restriction*. Firstly, when this argument is used without a restriction, beliefs about all of the elements known to the program are displayed. Secondly, when a restriction is applied, the display is limited to beliefs about the elements given by the collection represented in the *restriction*. Beliefs are displayed for element names in the order given by the ordering convention. See §6.2 for details of specifying collections for the definition part.

The first form of the syntax displays variances and covariances for all of the belief stores. The second form of the syntax limits the display to variances and covariances in belief store i.

f

Chapter 16

Exchangeable sequences

16.1 Introduction

Consider the collection $D = [D_1, \ldots, D_p]$, and suppose that we take a sample of size $n \geq 1$ over this collection. Let D_{it} be the t^{th} observation on D_i, and collect the observations $D_{1t}, \ldots, D_{pt}$ into the vector $D_{(t)}$. Collect all the quantities $D_{(1)}, \ldots, D_{(n)}$ together into the collection $C(n)$. Suppose that we might observe $D_{it} = d_{it}$. (Actual observations are not essential.) The following expectations and covariances summarise our requirements for second–order exchangeability:

$$\mathrm{E}(D_{it}) = m_i \text{ for all } t. \tag{16.1}$$

$$\mathrm{Cov}(D_{it}, D_{js}) = \begin{cases} g_{ij} + u_{ij} & \text{for } t = s \\ g_{ij} & \text{for } t \neq s \end{cases}. \tag{16.2}$$

Arrange the u_{ij} and g_{ij} as the entries of the $p \times p$ matrices U, G.

Such representations follow by explicitly writing the t^{th} observation on the i^{th} quantity

$$D_{it} = \mathcal{M}(D_i) + \mathcal{R}_t(D_i)$$

as the sum of a mean component plus a residual component, where the residual components $\mathcal{R}_t(\cdot)$ are uncorrelated with all other quantities. In principle, we can now specify

$$\begin{aligned} \mathrm{Cov}(\mathcal{M}(D_i), \mathcal{M}(D_j)) &= g_{ij} \\ \mathrm{Cov}(\mathcal{R}_t(D_i), \mathcal{R}_r(D_j)) &= \begin{cases} u_{ij} & \text{for } r = t, \\ 0 & \text{for } r \neq t. \end{cases} \end{aligned}$$

Then the variance matrix for the vector of averages

$$\bar{D} = \frac{1}{n} \sum_{r=1}^{n} D_r$$

is $\mathrm{Var}(\bar{D}) = G + \frac{1}{n} U$. This variance matrix is coherent if both G and U are nonnegative definite.

In the context of the specifications one might actually make, it is often more natural to specify G and $G+U$, rather than G and U. This follows because $\mathrm{Var}(D_{(r)}) = G+U$ is the variance of a typical member of the exchangeable sequence. It is for this reason that the commands involving exchangeable sequences typically demand that there are two belief stores, one containing G and the other $G+U$, pointed to by the modelvar and infovar controls respectively.

Various operators can be used to generate output related to exchangeable adjustments. See §21.4 for details.

16.2 Exploiting exchangeability

The commands in the remainder of this chapter require that a preliminary adjustment has been made, following which the commands here are used to determine the effects of changing the sample size. These commands may be used when the following conditions are observed.

1. That there are a collection of exchangeable quantities D_i, with properties as outlined in §16.1, which will be or have been used to form an adjustment.
2. That the collection of quantities being adjusted, $[B_1 \ldots B_k]$ must be second–order exchangeable with $C(n)$. That is, we must have

$$\mathrm{Cov}(B_j, D_{it}) = l_{ji} \text{ for all } t,$$

   Arrange the l_{ji} as the entries of the $k \times p$ matrix L.
3. An exchangeable adjustment [B/C(n)] must have taken place already, either with a sample size $n > 1$, or with the exchangeable control switched on.

16.3 Organising exchangeable adjustments

Suppose that our notation is as in §16.2, and that the quantities are second–order exchangeable as described. In addition to the usual factors concerning adjustments, the following come into play: whether or not there is any data; the source of the variances and covariances required; and the initial sample size used.

16.3.1 Source of variances and covariances required

Refer to §20.6. There are four sources to consider, the first three are as for any general adjustment. The source for $\mathrm{Var}(B)$ is set using the priorvar control. The source for $\mathrm{Cov}(B, D_{(t)}) = L$ is set by using the betweenvar control. The source for $\mathrm{Var}(D_{(i)}) = G + U$ is set by using the infovar control. In addition to these usual sources, we need also the source for the covariances between different observations, $\mathrm{Cov}(D_{(i)}, D_{(j)}) = G,\ \forall\, i, j$. This is set by using the modelvar control, and should typically be a source different to the one set by the infovar control. For example, suppose that $\mathrm{Var}(B)$, L, and $G + U$ have been entered into belief store number 1, and that G has been entered into belief store number 2. The appropriate command line defining the belief sources is

BD>**control priorvar=1,betweenvar=1,infovar=1,modelvar=2** ↩

16.3.2 Choosing the initial sample size

We show in Figure 16.1 what steps must be considered in choosing the sample size for a preliminary adjustment. In turn, these are:

1. The usedata control. When switched off, actual observations are ignored, and so a fictitious sample size must be set for the adjustment. Otherwise the sample size used is the same as the number of actual observations.
2. The obs control. This is used to set a fictitious sample size, whenever necessary and assuming the absence of (or ignoring) actual data.
3. The exchangeable control. This is used to force the use of exchangeable beliefs when the sample size, actual or fictitious, is $n = 1$. (By default, when $n = 1$ [B/D] assumes non-exchangeability for efficiency.)
4. The matchcd control. This is used to force the use of special algorithms which exploit analytical results to avoid much unnecessary calculation when the default algorithms are used to calculate the adjustments. The setting of the control is relevant only when a predictive or mean component analysis for a pure exchangeable system is taking place. In these circumstances, the adjustments

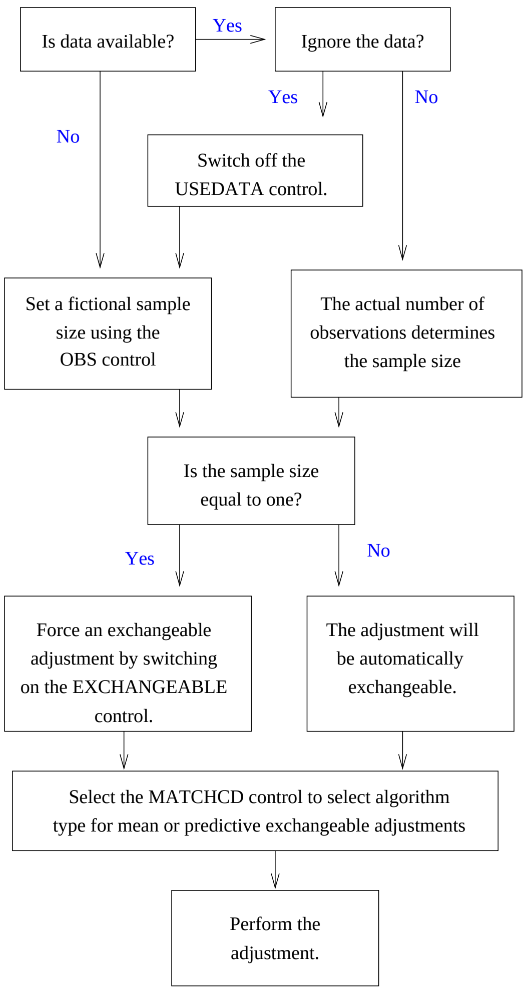
can be arranged so that the canonical directions for the mean and predictive adjustments are the same (except for normalising constants) in the case of rank degeneracy. (Otherwise, the canonical directions are the same anyway, except for normalising constants.) However, note that the setting of this control can, in the case of rank degeneracy, give canonical directions for the mean component adjustment which are valid but different (because of non-uniqueness) to the directions computed by the default algorithm. In particular, if you select the matchcd control, in the special case of mean component adjustment for an exchangeable system under rank degeneracy, the canonical directions reported for the resolution matrix are not generally the same as those reported when using the default algorithms. However, we recommend that you set the matchcd control to ON.

These controls are discussed in greater detail in §20.3. As an example, suppose that you wish to generate a preliminary adjustment of B by D (assume that these are also the names of the respective [B/D] *bases*) where you wish to take initially a sample of size $n = 1$, and where you wish to ignore the actual observations that you have on D. Then the following commands are appropriate:

BD>**control -usedata,obs=1,exchangeable** ↩

BD>**adjust [B/D]** ↩

Figure 16.1: Controls affecting exchangeable adjustments



16.4 Determining sample sizes to guarantee variance reductions

▷ ▷ **Syntax**

BD>**findsize :E** ↩

where E is any valid equation which results in a real positive number such that $0 < \mathrm{E} < 1$.

◁ ◁

The FINDSIZE: command is used to determine, by exploiting second-order exchangeability, the sample sizes required to guarantee requested levels of variance reduction in individual quantities and the system overall. Suppose that [B/D] currently holds the preliminary adjustment $[B/C(n)]$ as detailed in §16.3. The output consists of the current and maximal resolutions for each $B_j \in B$ and for B overall, together with sample sizes and extra sample sizes to guarantee two alternative degrees of increase in resolution.

For the collection B considered as a whole, a row is output consisting of the following:

1. the name of the collection, B;
2. the current resolution in the collection, $\mathrm{R}_{C(n)}(B)$;
3. the maximal resolution in the collection, $\phi = \lim_{n\to\infty} \mathrm{R}_{C(n)}(B)$;
4. the smallest sample size n_0 such that $\mathrm{R}_{C(n_0)}(B) > E$;
5. the smallest additional sample sise m_0 such that $\mathrm{R}_{C(n+m_0)}(B) > \mathrm{R}_{C(n)}(B) + E(\phi - \mathrm{R}_{C(n)}(B))$.

Then, for every element $B_j \in B$ (output can be restricted to some subset of elements using the OPTION: command), a row is output consisting of thhe following.

1. the name of the element, B_j;
2. the current resolution in the collection, $\mathrm{R}_{C(n)}(B_j)$;
3. the maximal resolution in the collection, $\phi_j = \lim_{n\to\infty} \mathrm{R}_{C(n)}(B_j)$;
4. the smallest sample size n_j such that $\mathrm{R}_{C(n_j)}(B_j) > E$;
5. the smallest extra sample size m_j such that $\mathrm{R}_{C(n+m_j)}(B_j) > \mathrm{R}_{C(n)}(B) + E(\phi_j - \mathrm{R}_{C(n)}(B_j))$.

Hence, the penultimate column of output consists of sample sizes unconditionally guaranteed to achieve minimum resolutions in uncertainty for the collection and its individual elements. [B/D] indicates whenever the degree of resolution required is not achievable. The final column relates to effects in addition to the current adjustment, and shows the extra sample sizes required to guarantee extra degrees of resolution.

The kind of unconditional output seen in the penultimate column can be accessed for both the collection and its component elements by using the **findsize** operator.

16.5 Exploring the effects of changing sample sizes

▷ ▷ **Syntax**

BD>**varysize: E1,E2,E3** ↩

where E1, E2, and E3 are any valid equations which result, on rounding, in positive integers I_1, I_2, I_3, *such that* $I_1 \leq I_3$ *and* $I_2 > 0$.

◁ ◁

The VARYSIZE: command is used for the following purpose. Whenever an exchangeable adjustment has taken place (that is, an adjustment satisfying the conditions of §16.2) then it is possible to explore the effect, on the current adjustment, of changing the sample sizes in the sense that some of the output generally available for an adjustment is generated for changed sample sizes by exploiting the underlying exchangeability. That is, changing the sample size generally leads to an adjustment quite different to the current one, but certain summary aspects of the altered adjustment may be obtained cheaply.

Assume that the initial adjustment was for [B/C(n)]. The effect of this command is as follows. Any output currently selected by the options summarised below is computed and given assuming the sequence of adjustments $[B/C(I_1)]$, $[B/C(I_1 + I_2)]$, and so forth; and finally for the largest integer m such that $I_1 + mI_2 \leq I_3$, $[B/C(I_1 + mI_2)]$. The options which are affected are as follows:

`bases` for titling, and to summarise the elements in the adjustment;

`ru` for a summary of the effect of the adjustment on the system resolution;

`v` for output of the adjusted variances;

`e` for output of the adjusted expectations;

`e*` for output of the standardised adjusted expectations;

`pv` for output of partial adjustment variances, if any.

Note that any comparative information yielded refers to comparisons between the current and previous adjustment. In the context of the VARYSIZE: command, results are output for different sample-sizes on effectively the same adjustment, and the previous adjustment remains unchanged.

Chapter 17

Plotting data

17.1 Labelling plots

Plots may be labelled as follows. A data-carrier can be associated with a labelling data carrier when the PLOT: or LOPLOT: command is issued. The labelling data carrier is just like any other data carrier, except that its values are integers in the range 0 to 55 inclusive. (Any values outside this range will produce the *missing symbol*, which can be redefined by using the missingsymb control.) These integers match the characters shown in Table 17.1.

Table 17.1: Characters used to label plots

0		14	N	28		42	n
1	A	15	O	29	a	43	o
2	B	16	P	30	b	44	p
3	C	17	Q	31	c	45	q
4	D	18	R	32	d	46	r
5	E	19	S	33	e	47	s
6	F	20	T	34	f	48	t
7	G	21	U	35	g	49	u
8	H	22	V	36	h	50	v
9	I	23	W	37	i	51	w
10	J	24	X	38	j	52	x
11	K	25	Y	39	k	53	y
12	L	26	Z	40	l	54	z
13	M	27	+	41	m	55	#

The effect will be that, for example, a plot of X versus Y where Y is associated with a labelling data carrier Z, will plot the symbol associated with Z_i for the point (X_i, Y_i). Thus, if $Z_i = 11$, the symbol 'K' will be plotted.

Notice that the possible set of labelling symbols is composed of two matching sets of 28 symbols, each consisting of a blank, the letters of the alphabet in upper or lower case, and a standard plotting character.

17.2 Low resolution plots

▹ ▹ **Syntax**

BD>**loplot: N1, B1** [=**L1**] [, **B2** [=**L2**]] . . . ↩

where $N1$ is the name of a data-carrier; $B1, B2, \ldots$ are the names of bases of data-carriers, or the names of single data-carriers; and $L1, L2, \ldots$ are the names of data-carriers.

◁ ◁

The LOPLOT: command is used to produce low-resolution plots. The command has at least two arguments. $N1$ is the name of the data carrier to appear on the x-axis. The remaining arguments $B1, B2, \ldots$ define a collection of quantities, each of which will appear on the y-axis. The result will be several superimposed plots of $N1$ versus each quantity in $B1, B2, \ldots$, all plotted to the same scale. (Clearly the quantities to appear simultaneously on the y-axis should be of similar magnitude, otherwise the superimposed plots will be cramped in different regions of the overall plot.)

Each data carrier to appear on the y-axis can be associated with a labelling vector L. Further, if a base name rather than the name of a data-carrier is associated with a labelling vector, then this labelling vector is associated with every data carrier in the base.

For low resolution plots, it is not possible to connect consecutively plotted points with a straight line; nor is it possible to disable the plotting of points. Consequently, the `connect` and `nopoints` options are ignored here.

It is possible to change the resolution of the low resolution plots via the plotlines and plotcols. These are, by default, 18 and 75 respectively, although they may be varied between 10 and 253; and 30 and 250 respectively. Each possible point location is a character point on the screen.

Up to 26 superimposed plots are allowed, although there may be insufficient space to report information about all of them on the plot, and the plot itself may be hard to interpret. For unlabelled plots, and where there are three or fewer superimposed plots, special characters are plotted for the different y-axis quantities. For four or more unlabelled plots, the letters of the alphabet A..Z are plotted for the y-axis quantities. If several points hit the same plotting address, the number of counts is plotted (a nine represents nine or more counts).

It is possible to plot with a single quantity appearing on the y-axis and up to 26 quantities appearing on the x-axis, instead. This will be the effect of the PLOT: command if the plotxyy control has been switched off. Otherwise the appearance of the plot is governed by the same considerations.

The amount of data plotted will depend on the number of cases for which observations exist on the single data-carrier to appear on the x-axis (or instead the y-axis if the plotxyy control has been switched off). If the autoselect control is switched on, then all possible cases will be included. Otherwise, if a data selection has been made using the SELECT: command, only the cases which have been selected *and* for which observations on the single data carrier exist will be included. Finally, any points corresponding to missing data are ignored.

As an example, consider the sequence of commands given in Figure 17.1. (See also the example given in Figure 17.2 for a high resolution plot, and the example shown in Figure 17.3 where we have a single y-axis quantity and multiple x-axis quantities.) Assume that $x, u1, u2, u3, u4, za, zb$ are all data carriers of length n. The initial commands deal with organising the amount of data to be plotted: only the cases for which the data carrier x has values less than 0.5 will be considered. The remaining commands construct a base ua to contain two data carriers $u1, u2$, and then construct a plot of the n pairs $(x, u1)$ labelled by za, superimposes a plot of the n pairs $(x, u2)$ also labelled by za, superimposes a plot of the n pairs $(x, u3)$ labelled by zb, and finally superimposes a plot of the n pairs $(x, u4)$ unlabelled. The title of the plot as as defined in the preceding TITLE: command.

Figure 17.1: Example code for a low resolution plot

```
BD>control: -autoselect ↩
BD>select: ↩
BD>select: x<0.5 ↩
BD>base: ua=u1, u2 ↩
BD>title: A labelled plot of (x,u1) ... (x,u4). ↩
BD>loplot: x, ua=za, u3=zb, u4 ↩
```

17.3 High resolution plots

▷ ▷ **Syntax**

BD>**plot: N1, B1 [=L1] [, B2 [=L2]] . . .** ↩

where $N1$ is the name of a data-carrier; $B1, B2, \ldots$ are the names of bases of data-carriers, or the names of single data-carriers; and $L1, L2, \ldots$ are the names of data-carriers.

◁ ◁

The PLOT: command is used to produce high-resolution plots in a graphics window. The resolution of this window can be changed using the winxsize and winysize controls of §20.12. The command has at least two arguments. $N1$ is the name of the data carrier to appear on the x-axis. The remaining arguments $B1, B2, \ldots$ define a collection of quantities, each of which will appear on the y-axis. The result will be several superimposed plots of $N1$ versus each quantity in $B1, B2, \ldots$, all plotted to the same scale. (Clearly the quantities to appear simultaneously on the y-axis should be of similar magnitude, otherwise the superimposed plots will be cramped in different regions of the overall plot.)

Each data carrier to appear on the y-axis can be associated with a labelling vector L. Further, if a base name rather than the name of a data-carrier is associated with a labelling vector, then this labelling vector is associated with every data carrier in the base.

Consecutive points plotted will be connected by a line if the `connect` option is switched on. If the `nopoints` option is also switched on, then no points, and only these connecting lines will be drawn.

The scale of the plot is, by default, the appropriate range of x and y values being plotted. The scaling may be changed by using the xscale and yscale controls. For example, suppose that the actual x and y ranges are $x \in (0, 1)$, $yin(0, 1)$. Then the entire plot region will be given over to the (0,0) to (1,1) square. However, we might wish the plot to be restricted to the top right hand quarter, say, and issue the command

BD>**control: xscale=-1,1, yscale=-1,1** ↩

Each axis is labelled according to the scales used, either default or otherwise. The number of decimal places used is 4 by default, but this can be changed by using the xaxisdp and yaxisdp controls, as in the following example:

BD>**control: xaxisdp=3, yaxisdp=0** ↩

Up to 26 superimposed plots are allowed, although there may be insufficient space to report information about all of them on the plot, and the plot itself may be hard to interpret. For unlabelled plots, and where there are three or fewer superimposed plots, special characters (a circle, a cross, a rectangle) are plotted for the different y-axis quantities. For four or more unlabelled plots, the letters of the alphabet A..Z are plotted for the y-axis quantities. For colour screens, different combinations of colours and line styles are used to distinguish between different plots.

It is possible to plot with a single quantity appearing on the y-axis and up to 26 quantities appearing on the x-axis, instead. This will be the effect of the PLOT: command if the plotxyy control has been switched off. Otherwise the appearance of the plot is governed by the same considerations.

The amount of data plotted will depend on the number of cases for which observations exist on the single data-carrier to appear on the x-axis (or instead the y-axis if the plotxyy control has been switched off). If the autoselect control is switched on, then all possible cases will be included. Otherwise, if a data selection has been made using the SELECT: command, only the cases which have been selected *and* for which observations on the single data carrier exist will be included. Finally, any points corresponding to missing data are ignored.

As an example, consider the sequence of commands given in Figure 17.2. (See also the example given in Figure 17.1 for a labelled low resolution plot.) Assume that $x, u1, u2, u3, u4$ are all data carriers of length n. The initial command ensures that all cases for which the data carrier x has values will be included. The remaining commands construct a base ua to contain two data carriers $u1, u2$, and then construct a plot of the n pairs $(x, u1)$, superimposes a plot of the n pairs $(x, u2)$, superimposes a plot of the n pairs $(x, u3)$, and finally superimposes a plot of the n pairs $(x, u4)$. The `connect` and `nopoints` options determine that the actual points are not plotted, but joined together with a straight line. The title of the plot as as defined in the preceding TITLE: command.

For an example where a single y-axis data carrier y is plotted against many x-axis quantities, see Figure 17.3 and note the use of the plotxyy control.

Figure 17.2: Example code for a high resolution plot

```
BD>control: autoselect ↩
BD>base: ua=u1, u2 ↩
BD>title: An unlabelled plot of (x,u1) ... (x,u4). ↩
BD>option: connect, nopoints ↩
BD>plot: x, ua, u3, u4 ↩
```

Figure 17.3: Example code for multiple x-axis quantities

```
BD>control: autoselect, -plotxyy ↩
BD>base: ua=u1, u2 ↩
BD>title: An unlabelled plot of (u1,y) ... (u4,y). ↩
BD>plot: y, ua, u3, u4 ↩
```

17.4 Defining titles for plots

▹ ▹ **Syntax**

1. BD>**title: T** ↩

2. BD>**title:** ↩

where T is any sequence of alphanumeric text.

◃ ◃

The TITLE: command is used to define a title to be used on both low and high resolution plots. As for the STRING:, ITITLE: and PRINT: commands, items in square brackets, [...], are not parsed for substitutions, and the text does not have its blank spaces removed, or its capital letters changed to lower case.

The second form of the syntax, where the command is given without an argument, deletes any such title.

Chapter 18

Interactively retaining output

18.1 Introduction

The KEEP: command is used to force [B/D] to retain various kinds of output in user–defined quantities. The general syntax of the command is as follows.

▹ ▹ **Syntax**

BD>**keep: [-]Argument1** [,[-]**Argument2**] [, ...] ↩

where Argument1, Argument2, ... *are arguments of the form discussed below. The preceding of an* Argument *by the symbol "-" means that the retention of the indicated output is to cease.*

◃ ◃

The KEEP: command is used to force [B/D] to retain various kinds of output in user–defined quantities. A list of the possible arguments is also available by issuing the LOOK: command with argument keep; and a list of currently kept items is available by using the LOOK: command with the kept argument.

18.2 Retaining output from an iterative adjustment

The following possibilities relate only to the iterative adjustments available by issuing the ITADJUST: command. Suppose that the base B is to be iteratively adjusted by the quantities $X_1, \ldots, X_n$. Suppose that we define the collection $X_{(i)} = X_1, \ldots, X_i$.

Expected size of an iterative adjustment

▹ ▹ **Syntax**

1. BD>**keep: itesize=D** ↩

2. BD>**keep: -itesize** ↩

where D is the name of a data–carrier.

◃ ◃

The first form of the syntax is used to give the name of a data–carrier which is to be used for storing consecutive resolution transform traces. Whenever the iterative adjustment is carried out, the i^{th} observation on the data–carrier D becomes defined as $\text{trace}(\text{T}_{[X_{(i)}]}(B))$. These values are equal to the expected sizes of the adjustment: $\text{E}(\text{Size}_{X_{(i)}}(B))$. The actual adjustment sizes are available via the itsize argument below, and the consecutive size ratios may be obtained by dividing the latter by the former.

The second form is used to switch off the automatic storing of the resolution transform traces. Any data–carrier defined beforehand remains defined together with its contents.

Path correlations within an iterative adjustment

▷ ▷ **Syntax**

1. BD>**keep: itpath=D** ↩

2. BD>**keep: -itpath** ↩

where D is the name of a data–carrier.

◁ ◁

The first form of the syntax is used to give the name of a data–carrier which is to be used for storing consecutive path correlations. Whenever the iterative adjustment is carried out, the i^{th} observation on the data–carrier D becomes defined as $\mathrm{C}(X_{(i-1)}, X_i)$. The first observation, for which there is no path correlation, is defined as zero.

The second form is used to switch off the automatic storing of the path correlation Any data–carrier defined beforehand remains defined together with its contents.

Observed size of an iterative adjustment

▷ ▷ **Syntax**

1. BD>**keep: itsize=D** ↩

2. BD>**keep: -itsize** ↩

where D is the name of a data–carrier.

◁ ◁

The first form of the syntax is used to give the name of a data–carrier which is to be used for storing consecutive sizes for the adjustments. Whenever the iterative adjustment is carried out, the i^{th} observation on the data–carrier D becomes defined as $\mathrm{Size}_{x_{(i)}}(B)$, where $x_{(i)}$ is the observed value of $X_{(i)}$. The expected size of the adjustment can be obtained via the itesize argument discussed above, and the former divided by the latter yields the corresponding size ratio for the adjustment.

The second form is used to switch off the automatic storing of the size of the adjustment. Any data–carrier defined beforehand remains defined together with its contents.

18.3 Retaining from standard adjustments

Consider the following scenario: Suppose that the *base* $\{B\}$ has been adjusted by the *base* $\{D\}$. Occasionally we will be concerned with partial adjustments, and to this purpose for the partial adjustment results we suppose that we adjust initially by the *base* $\{E\}$, and then partially by *base* $\{F\}$; and that D represents the complete adjustment by the *base* $D = E \cup F$. Suppose that the notional sample size is n potential observations on each *element* of D.

Bearings

▷ ▷ **Syntax**

1. BD>**keep: b=A** ↩

2. BD>**keep: -b** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the bearing during an adjustment. The bearing is $\mathrm{Z}_d(B)$, a linear combination of the elements of B. This linear combination is retained as an assignment, using the supplied name "A", which may not include the character ".". The constant part of the adjusted expectation is included in the stored assignment.

The switch remains on until switched off. Consequently, care must be taken not to overwrite a bearing stored as an assignment by ones emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the bearing as an assignment. Any assignment so defined during previous adjustments remain defined.

Partial bearings

▷ ▷ **Syntax**

1. `BD>`**keep: pb=A** ↩

2. `BD>`**keep: -pb** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the partial bearing during an adjustment. Assuming the adjustment to be of B by D and then F, the partial bearing is $\mathrm{Z}_{f/d}(B)$, a linear combination of the elements of B. This linear combination is retained as an assignment, using the supplied name "A", which may not include the character ".". The constant part of the adjusted expectation is included in the stored assignment.

The switch remains on until switched off. Consequently, care must be taken not to overwrite a bearing stored as an assignment by ones emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the bearing as an assignment. Any assignment so defined during previous adjustments remain defined.

Discrepancy vectors

▷ ▷ **Syntax**

1. `BD>`**keep: dv=A** ↩

2. `BD>`**keep: -dv** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the discrepancy vector during an adjustment. The discrepancy vector is $\mathrm{Dis}(d)\mathrm{B}$, a linear combination of the elements of B. This linear combination is retained as an assignment, using the supplied name "A", which may not include the character ".". The constant part of the adjusted expectation is included in the stored assignment.

The switch remains on until switched off. Consequently, care must be taken not to overwrite a discrepancy vector stored as an assignment by ones emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the discrepancy vector as an assignment. Any assignment so defined during previous adjustments remain defined.

Partial discrepancy vectors

▷ ▷ **Syntax**

1. BD>**keep: pdv=A** ↩

2. BD>**keep: -pdv** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the partial discrepancy vector during an adjustment. Assuming the adjustment to be of B by D and then F, the partial discrepancy vector is Dis(f/d)B, a linear combination of the elements of B. This linear combination is retained as an assignment, using the supplied name "A", which may not include the character ".". The constant part of the adjusted expectation is included in the stored assignment.

The switch remains on until switched off. Consequently, care must be taken not to overwrite a discrepancy vector stored as an assignment by ones emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the discrepancy vector as an assignment. Any assignment so defined during previous adjustments remain defined.

Canonical directions

▷ ▷ **Syntax**

1. BD>**keep: cd=A** ↩

2. BD>**keep: -cd** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of canonical directions during an adjustment. Suppose that an adjustment results in a resolution transform whose canonical directions are $W_1, \dots, W_\tau$. We restrict attention to canonical directions corresponding to positive canonical resolutions. The number of these, τ, is available via the operand **rmrank** discussed in §21.3. Each canonical direction is a linear combination of the component elements of the base B being adjusted, $B_1, \dots, B_k$. These linear combinations are retained as assignments, using the supplied name "A" as a stem. W_1 will be stored as the assignment whose name is $A1$, and so forth, W_τ being stored as the assignment whose name is "Aτ". The corresponding canonical resolutions are available as the [B/D] operators **cr**(1) ... **cr**(τ). The constant part of the canonical direction is included in the stored assignment. The name "A" supplied must not include a "." character.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the canonical directions that were stored as assignments by those emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the canonical directions as assignments. Any assignments so defined during previous adjustments remain defined.

Adjusted expectations

▷ ▷ **Syntax**

1. BD>**keep: e=A** ↩

2. BD>**keep: -e** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of adjusted expectations during an adjustment. Suppose that the component elements of the base B being adjusted are $B_1, \ldots, B_k$. Then the adjusted expectation for B_k is $\mathrm{E}_D(B_k)$, a linear combination of the component elements $D_1, \ldots, D_r$ of the adjusting base D. These linear combinations are retained as assignments, using the supplied name "A" as a stem. $\mathrm{E}_D(B_1)$ will be stored as the assignment whose name is $A1$, and so forth, $\mathrm{E}_D(B_9)$ being stored as the assignment whose name is "A9". The constant part of the adjusted expectation is included in the stored assignment. The name "A" supplied must not include a "." character.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the adjusted expectations that were stored as assignments by those emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the adjusted expectations as assignments. Any assignments so defined during previous adjustments remain defined.

The storing of adjusted expectations as assignments is not available for iterative adjustments.

Name of the adjusting base

▷ ▷ **Syntax**

1. `BD>`**keep: fitted=N** ↩

2. `BD>`**keep: -fitted** ↩

where N is any valid name*, intended to become the name of a* base*.*

◁ ◁

The first form of the syntax is used to retain, using the name supplied, the currently fitted *base*. Hence, if the current adjustment can be referred to as $[B/D_1 + D_2 + \ldots + D_r]$, then N will become the base consisting of the bases and elements $\{D_1, \ldots, D_r\}$. This command has immediate effect if there exists a current adjustment, so that one need not not wait for a forthcoming adjust-type command. Every new adjustment causes the redefinition of this base, and the base name N used can be such that a former base of the same name is overwritten. The adjusted argument to the KEEP: command is similar, but applies to the base being adjusted. The control applies only to the ADJUST: command.

The second form of the command discontinues retention of the currently fitted base.

Name of the adjusted base

▷ ▷ **Syntax**

1. `BD>`**keep: adjusted=N** ↩

2. `BD>`**keep: -adjusted** ↩

where N is any valid name*, intended to become the name of a* base*.*

◁ ◁

The first form of the syntax is used to retain, using the name supplied, the currently adjusted *base*. Hence, if the current adjustment can be referred to as $[B/D]$, then N will become the base consisting of the base B. This command has immediate effect if there exists a current adjustment, so that one need not not wait for a forthcoming adjust-type command. Every new adjustment causes the redefinition of this base, and the base name N used can be such that a former base of the same name is overwritten. The fitted argument to the KEEP: command is similar, but applies to the base used for adjusting. The control applies only to the ADJUST: command.

The second form of the command discontinues retention of the currently adjusted base.

Partial canonical directions

▷ ▷ **Syntax**

1. BD>**keep: pcd=A** ↩

2. BD>**keep: -pcd** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of partial canonical directions during an adjustment. Suppose that a partial adjustment results in a partial resolution transform whose partial canonical directions are $Z_1, \ldots, Z_\tau$. We restrict attention to partial canonical directions corresponding to positive partial canonical resolutions. The number of these, τ, is available via the operand **prmrank** discussed in §21.3. Each partial canonical direction is a linear combination of the component elements of the base B being adjusted, $B_1, \ldots, B_k$. These linear combinations are retained as assignments, using the supplied name "A" as a stem. Z_1 will be stored as the assignment whose name is $A1$, and so forth, Z_τ being stored as the assignment whose name is "Aτ". The corresponding partial canonical resolutions are available as the [B/D] operators **pcr**(1) ... **pcr**(τ). The constant part of the canonical direction is included in the stored assignment. The name "A" supplied must not include a "." character.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the partial canonical directions that were stored as assignments by those emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the partial canonical directions as assignments. Any assignments so defined during previous adjustments remain defined.

Resolution matrix columns

▷ ▷ **Syntax**

1. BD>**keep: rm=A** ↩

2. BD>**keep: -rm** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the columns of the resolution matrix, $\mathrm{Var}(B)^\dagger\mathrm{Cov}(B,D)\mathrm{Var}(D)^\dagger\mathrm{Cov}(D,B)$, as assignments during an adjustment. Each column is interpreted as a linear combination of the component elements of the base B being adjusted, $B_1, \ldots, B_k$. These linear combinations are retained as assignments, using the supplied name "A" as a stem. The first column will be stored as the assignment whose name is $A1$, and so forth. No constant part is included in the stored assignment. The name "A" supplied must not include a "." character.

The coefficients making up the resolution matrix are available interactively by using the **ascf** function. These numbers reproduce the output available under the `rm` option.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the columns that were stored as assignments by those emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the resolution matrix columns as assignments. Any assignments so defined during previous adjustments remain defined.

Maximal resolution matrix columns

▷ ▷ **Syntax**

1. BD>**keep: mrm=A** ↩

2. BD>**keep: -mrm** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the columns of the maximal resolution matrix, $\lim n \to \infty \mathrm{Var}(B)^\dagger \mathrm{Cov}(B,D)\mathrm{Var}(D)^\dagger \mathrm{Cov}(D,B)$, as assignments during an adjustment. Each column is interpreted as a linear combination of the component elements of the base B being adjusted, $B_1,\ldots,B_k$. These linear combinations are retained as assignments, using the supplied name "A" as a stem. The first column will be stored as the assignment whose name is $A1$, and so forth. No constant part is included in the stored assignment. The name "A" supplied must not include a "." character.

The coefficients making up the maximal resolution matrix are available interactively by using the **ascf** function. These numbers reproduce the output available under the `mrm` option.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the columns that were stored as assignments by those emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the maximal resolution matrix columns as assignments. Any assignments so defined during previous adjustments remain defined.

Partial resolution matrix columns

▷ ▷ **Syntax**

1. BD>**keep: prm=A** ↩

2. BD>**keep: -prm** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the columns of the partial resolution matrix,

$$(\mathrm{Var}(B))^\dagger[\mathrm{Cov}(B,D^T)(\mathrm{Var}(D))^\dagger\mathrm{Cov}(D,B^T) - \mathrm{Cov}(B,E^T)(\mathrm{Var}(E))^\dagger\mathrm{Cov}(E,B^T)],$$

as assignments during an adjustment. Each column is interpreted as a linear combination of the component elements of the base B being adjusted, $B_1,\ldots,B_k$. These linear combinations are retained as assignments, using the supplied name "A" as a stem. The first column will be stored as the assignment whose name is $A1$, and so forth. No constant part is included in the stored assignment. The name "A" supplied must not include a "." character.

The coefficients making up the partial resolution matrix are available interactively by using the **ascf** function. These numbers reproduce the output available under the `prm` option.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the columns that were stored as assignments by those emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the partial resolution matrix columns as assignments. Any assignments so defined during previous adjustments remain defined.

Adjusted versions

▷ ▷ **Syntax**

1. BD>**keep: av=A** ↩

2. BD>**keep: -av** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of adjusted versions during an adjustment. Corresponding to each element B_i is an adjusted version $B_i - \mathrm{E}_D(B_i)$. Each adjusted version is a linear combination of the component elements of the base D being used to adjust B.

These linear combinations are retained as assignments, using the supplied name "A" as a stem. Thus, $B_1 - \mathrm{E}_D(B_1)$ will be stored as the assignment whose name is $A1$, and so forth. The constant part of every adjusted version is zero. The name "A" supplied must not include a "." character.

Be warned that there are subtleties involved in building adjusted versions retained from general or pure exchangeable adjustments; the direct building of the assignments won't produce elements with the beliefs that you might have intended because the assignments "forget" about the exchangeable structure of the information source elements.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the adjusted versions that were stored as assignments by those emanating from fresh adjustments, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the adsjusted versions as assignments. Any assignments so defined during previous adjustments remain defined.

18.4 Recording input and output

Record screen output

▷ ▷ **Syntax**

1. BD>**keep: log=F** ↩

2. BD>**keep: -log** ↩

where F is the name of a file.

◁ ◁

The first form of the syntax is used to switch on the recording of output into the file name F supplied. The file will be written to the current directory. If the file already exists, it is overwritten. If the recording of output is already switched on and directed to a different file, this file is closed down and retained.

The second form of the syntax is used to turn off the recording of output. The associated file is closed and retained.

Output essentially refers only to the [B/D] output that has been selected by using the OPTION: command. Output is flushed to file (or screen and log file, if any) at the end of each command. If a crash happens during a command, any output from that command will be lost. Output redirected from screen to file is not reproduced on the log file.

Record keyboard input

▷ ▷ **Syntax**

1. BD>**keep: history=F** ↩

2. BD>**keep: -history** ↩

where F is the name of a file.

◁ ◁

The first form of the syntax is used to switch on the recording of keyboard input via the command history into the history file given by the name F supplied. The file will be written to the current directory. If the file already exists, it is overwritten. If the recording of keyboard input is already switched on and directed to a different file, this file is closed down and retained.

The second form of the syntax is used to turn off the recording of keyboard input. The associated file is closed and retained.

The individual commands and so forth which constitute the keyboard input are output at regular intervals: whenever the list of recorded items becomes full, the older half of the list is dumped to file. The total number of records in the list is fixed - its value can be seen if you issue a LOOK: command with the program argument. The list is fully flushed to the file whenever the file is closed down using the second form of the syntax and also when the program is terminated. Note that the delay in dumping records to file allows limited editing of the recorded items. See, for example, the /: command.

The command line which switches on or off recording is not recorded, and nor is the command which stops the program.

18.5 Other retentions

Belief comparison canonical directions

▷ ▷ **Syntax**

1. BD>**keep: bcd=A** ↩

2. BD>**keep: -bcd** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the generalised eigenvectors (canonical directions) resulting from a comparison of belief hypotheses about a collection of quantities, X, by using the COMPARE: command. These generalised eigenvectors are linear combinations, $a_1^T X, \ldots$, of the quantities in the collection. The corresponding pair of eigenvalues is available via the **bcd1** and **bcd2** operators.

These linear combinations are retained as assignments, using the supplied name "A" as a stem. $a_1^T X$ will be stored as the assignment whose name is $A1$, $a_2^T X$ as the assignment whose name is "A2", and so forth. The constant part of the canonical direction is included in the stored assignment. The name "A" supplied must not include a "." character.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the generalised eigenvectors stored as assignments by ones emanating from fresh belief comparisons, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the generalised eigenvectors as assignments. Any assignments so defined during previous belief comparisons remain defined.

Eigenvectors

▷ ▷ **Syntax**

1. BD>**keep: eig=A** ↩

2. BD>**keep: -eig** ↩

where A is the name of an assignment.

◁ ◁

The first form of the syntax is used to switch on the retention of the eigenvectors resulting from the computation of the eigenstructure implied by using the EIGEN: and INVERT: commands for beliefs specified over a collection of quantities, X. The eigenvectors are linear combinations, $a_1^T X, \ldots$, of the quantities in the collection.

These linear combinations are retained as assignments, using the supplied name "A" as a stem. $a_1^T X$ will be stored as the assignment whose name is $A1$, $a_2^T X$ as the assignment whose name is "A2", and so forth. No constant part is included in the stored assignment. The name "A" supplied must not include a "." character.

The switch remains on until switched off. Consequently, care must be taken not to overwrite the eigenvectors stored as assignments by ones emanating from fresh calculations of eigenstructure using the INVERT: and EIGEN: commands, unless this is what is desired.

The second form of the syntax is used to switch off the automatic storing of the eigenvectors as assignments. Any assignments so defined during previous calculations of eigenstructures remain defined.

Chapter 19

Output options

19.1 Introduction

Most of [B/D]'s output is controlled by the output options described in this chapter. Output options can be handled in two separate ways. Firstly the OPTION: command can be used to toggle an output option on or off, and the setting of the option remains in force until it is explicitly altered by a subsequent OPTION: command. We might use this way of setting an output option if we are interested in seeing the same output for (for example) repeated adjustments.

Secondly, the SHOW: command can be used to access the same output directly and temporarily. We might use this method to view the results of one adjustment before proceeding to other aspects of an analysis. Note that the temporary use of an option within a SHOW: command cannot affect the usual setting of the option.

Initially, all output options are switched off by default. A list of output options is available interactively by issuing a LOOK: command with argument `options`.

Many options come in standard, extended, and standardised form, such as the `cd`, `cd+`, and `cd*` options. We tend to use the '+' suffix to mean that the output will be more detailed than for the standard option; and the '*' suffix for standardised output. The latter tend to be concerned with linear combinations of quantities, and the standardisation usually means that the linear combinations are output with coefficients for quantities standardised to have expectation zero and variance unity.

19.2 Setting options

▹ ▹ **Syntax**

1. `BD>`**option:** [-]**Argument1** [,[-]**Argument2**] [, ...] ([**N1, N2,** ...]) ↩

2. `BD>`**option:** [-]**Argument1** [,[-]**Argument2**] [, ...] ↩

where Argument1, Argument2, ... *are arguments of the form discussed below; and where* $N1$, $N2, \ldots,$ *is a list of names of* elements *and* bases *in parenthesis and separated by commas.*

◃ ◃

The first form of the syntax for the OPTION: command is used to set a number of ouput options to on or off, depending on whether the option is prefaced by a minus sign (to switch off) or not. All options are off by default: Options generally govern the type and quantity of output given by the program, so that the default is minimal output.

The list of names of elements and bases $N1$, $N2, \ldots,$ defines a collection of elements to which output is to be restricted. Output is, in any case, only available for the elements in the collection being adjusted. See §6.2 for details of specifying collections for the definition part. The second form of the syntax gives output for all elements in the collection being adjusted.

Both the output options and the list of elements to which output is to be restricted remain in force until redefined by subsequent OPTION: commands. The SHOW: command, whose syntax is identical, can be used to specify one-off output options and element restrictions, but independently of the settings specified by OPTION: commands.

A list of possible options is available by issuing the LOOK: command with argument options.

19.3 Accessing the results of adjustments and partial adjustments

Consider the following scenario: Suppose that the *base* $\{B\}$ has been adjusted by the *base* $\{D\}$. Occasionally we will be concerned with partial adjustments, and to this purpose for the partial adjustment results we suppose that we adjust initially by the *base* $\{E\}$, and then partially by *base* $\{F\}$; and that D represents the complete adjustment by the *base* $D = E \cup F$. Suppose that the notional sample size is n potential observations on each *element* of D.

Canonical directions

▷ ▷ **Usage**

- BD>**option cd** ↩ *switches on standard canonical direction output.*
- BD>**option cd+** ↩ *switches on extended canonical direction output.*
- BD>**option -cd** ↩ *switches off canonical direction output.*
- BD>**show (cd)** ↩ *displays standard canonical direction output for the current adjustment.*
- BD>**show (cd+)** ↩ *displays extended canonical direction output for the current adjustment.*

◁ ◁

The cd and cd+ options yield the canonical resolutions and directions for the current adjustment. (See [31][section 3.6].) The standard output consists of the resolutions only; and the extended output of the resolutions with the corresponding directions represented as linear combinations of the *elements* of B. The directions are uncorrelated and are scaled so that each has prior variance unity. The option is ignored when there is no adjustment.

Standardised canonical directions

▷ ▷ **Usage**

- BD>**option cd*** ↩ *switches on standardised canonical direction output.*
- BD>**option -cd*** ↩ *switches off standardised canonical direction output.*
- BD>**show (cd*)** ↩ *displays standardised canonical direction output for the current adjustment.*

◁ ◁

The cd* option yields the standardised canonical resolutions and directions for the current adjustment. The directions are ouput as linear combinations of the *elements* of B, standardised such that each element has prior expectation zero and prior variance unity. The directions are uncorrelated and are scaled so that each direction has prior variance unity. The option is ignored when there is no adjustment.

Adjusted expectations

▷ ▷ **Usage**

- BD>**option e** ↩ *switches on adjusted expectation output.*
- BD>**option -e** ↩ *switches off adjusted expectation output.*
- BD>**show (e)** ↩ *displays adjusted expectation output for the current adjustment.*

◁ ◁

The e option is used to display the adjusted expectation for every element $X \in B$ for the current adjustment, $\mathrm{E}_D(X)$. It is shown as a linear combination of the *elements* of the *base* $\{D\}$. (See [31][sections 3.1, 3.2].) It is one of the options usable with the VARYSIZE: command. The e* option provides the same output in standardised form.

Standardised adjusted expectations

▷ ▷ **Usage**

- BD>**option e*** ↩ *switches on standardised adjusted expectation output.*
- BD>**option -e*** ↩ *switches off standardised adjusted expectation output.*
- BD>**show (e*)** ↩ *displays standardised adjusted expectation output for the current adjustment.*

◁ ◁

The e* option is used to display the standardised adjusted expectation for every *element* $X \in B$ for the current adjustment, $\mathrm{E}_D(X)$. It is shown as a linear combination of the standardised *elements* of the *base* $\{D\}$, standardised so that every *element* $D_i \in D$ has prior expectation zero and variance unity. (See [31][sections 3.1, 3.2].) It is one of the options usable with the VARYSIZE: command. The e option provides similar output.

Adjusted expectations for canonical quantities

▷ ▷ **Usage**

- BD>**option cde** ↩ *switches on adjusted expectation output for the canonical quantities.*
- BD>**option -cde** ↩ *switches off adjusted expectation output for the canonical quantities.*
- BD>**show (cde)** ↩ *displays adjusted expectation output for the canonical quantities of the current adjustment.*

◁ ◁

The cde option is used to display the adjusted expectation for every canonical quantity for the current adjustment. It is shown as a linear combination of the *elements* of the *base* $\{D\}$. The cde* option provides the same output in standardised form.

Standardised adjusted expectations for the canonical quantities

▷ ▷ **Usage**

- BD>**option cde*** ↩ *switches on standardised adjusted expectation output for the canonical quantities.*

- BD>**option -cde*** ↩ *switches off standardised adjusted expectation output for the canonical quantities.*

- BD>**show (cde*)** ↩ *displays standardised adjusted expectation output for the canonical quantities of the current adjustment.*

◁ ◁

The cde* option is used to display the standardised adjusted expectation for the canonical quantities for the current adjustment, $\mathrm{E}_D(X)$. It is shown as a linear combination of the standardised *elements* of the *base* $\{D\}$, standardised so that every *element* $D_i \in D$ has prior expectation zero and variance unity. The cde option provides similar output.

Bases subject to the adjustment

▷ ▷ **Usage**

- BD>**option bases** ↩ *enable the display of base titling for adjustments.*

- BD>**option -bases** ↩ *disable the display of base titling for adjustments.*

- BD>**show (bases)** ↩ *displays the names of the* elements *and* bases *involved in the current adjustment.*

◁ ◁

The bases option is used to provide titling and an indication of the *base* being adjusted and the current fit. It is one of the options usable with the VARYSIZE: command.

Maximal canonical directions

▷ ▷ **Usage**

- BD>**option mcd** ↩ *switches on standard maximal canonical direction output.*

- BD>**option mcd+** ↩ *switches on extended maximal canonical direction output.*

- BD>**option -mcd** ↩ *switches off maximal canonical direction output.*

- BD>**show (mcd)** ↩ *displays standard maximal canonical direction output for the current adjustment.*

- BD>**show (mcd+)** ↩ *displays extended maximal canonical direction output for the current adjustment.*

◁ ◁

The mcd and mcd+ options yield the maximal canonical resolutions and directions for the current maximal adjustment. The standard output consists of the resolutions only; and the extended output of the resolutions with the corresponding directions represented as linear combinations of the *elements* of B. The directions are uncorrelated and are scaled so that each has prior variance unity. The option is ignored when there is no maximal adjustment.

Standardised maximal canonical directions

▷ ▷ **Usage**

- BD>**option mcd*** ↩ *switches on standardised maximal canonical direction output.*
- BD>**option -mcd*** ↩ *switches off standardised maximal canonical direction output.*
- BD>**show (mcd*)** ↩ *displays standardised maximal canonical direction output for the current adjustment.*

◁ ◁

The mcd* option yields the standardised maximal canonical resolutions and directions for the current adjustment. The directions are ouput as linear combinations of the *elements* of B, standardised such that each element has prior expectation zero and prior variance unity. The directions are uncorrelated and are scaled so that each direction has prior variance unity. The option is ignored when there is no adjustment.

Maximal resolution matrix

▷ ▷ **Usage**

- BD>**option mrm** ↩ *switches on maximal resolution matrix output.*
- BD>**option -mrm** ↩ *switches off maximal resolution matrix output.*
- BD>**show (mrm)** ↩ *displays the maximal resolution matrix for the current adjustment.*

◁ ◁

The mrm option is available in the case of exchangeability to display the maximal resolution matrix for the full adjustment. For the full rank case, this is equal to

$$\lim_{n\to\infty}(\mathrm{Var}(B))^\dagger\mathrm{Cov}(B,D^T)(\mathrm{Var}(D))^\dagger\mathrm{Cov}(D,B^T).$$

Partial canonical directions

▷ ▷ **Usage**

- BD>**option pcd** ↩ *switches on standard partial canonical direction output.*
- BD>**option pcd+** ↩ *switches on extended partial canonical direction output.*
- BD>**option -pcd** ↩ *switches off partial canonical direction output.*
- BD>**show (pcd)** ↩ *displays standard partial canonical direction output for the current adjustment.*
- BD>**show (pcd+)** ↩ *displays extended partial canonical direction output for the current adjustment.*

◁ ◁

The pcd and pcd+ options yield the partial canonical resolutions and directions for the current partial adjustment. (See [31][section 5.3].) The standard output consists of the resolutions only; and the extended output of the resolutions with the corresponding directions represented as linear combinations of the *elements* of B. The directions are uncorrelated and are scaled so that each has prior variance unity. The option is ignored when there is no partial adjustment.

Standardised partial canonical directions

▷ ▷ **Usage**

- BD>**option pcd*** ↩ *switches on standardised partial canonical direction output.*
- BD>**option -pcd*** ↩ *switches off standardised partial canonical direction output.*
- BD>**show (pcd*)** ↩ *displays standardised partial canonical direction output for the current adjustment.*

◁ ◁

The pcd* option yields the standardised partial canonical resolutions and directions for the current adjustment. The directions are ouput as linear combinations of the *elements* of B, standardised such that each element has prior expectation zero and prior variance unity. The directions are uncorrelated and are scaled so that each direction has prior variance unity. The option is ignored when there is no adjustment.

Partial resolution matrix

▷ ▷ **Usage**

- BD>**option prm** ↩ *switches on partial resolution matrix output.*
- BD>**option -prm** ↩ *switches off partial resolution matrix output.*
- BD>**show (prm)** ↩ *displays the partial resolution matrix for the current partial adjustment.*

◁ ◁

The prm option is used to display the partial resolution matrix for a partial adjustment. For the full rank case, this is equal to

$$(\mathrm{Var}(B))^{\dagger}[\mathrm{Cov}(B,D^T)(\mathrm{Var}(D))^{\dagger}\mathrm{Cov}(D,B^T)-\mathrm{Cov}(B,E^T)(\mathrm{Var}(E))^{\dagger}\mathrm{Cov}(E,B^T)].$$

Variances for a partial adjustment

▷ ▷ **Usage**

- BD>**option pv** ↩ *switches on partial adjustment variance output.*
- BD>**option -pv** ↩ *switches off partial adjustment variance output.*
- BD>**show (pv)** ↩ *displays partial adjustment variance output for the current adjustment.*

◁ ◁

The pv option is only available when there is both a previous and a current adjustment, and is not available for the iterative algorithm. It displays the effects of the partial adjustment on several variance and uncertainty quantities. It is one of the options usable with the VARYSIZE: command.

For every *element* $X \in B$ the output consists of the *element*'s name X; the adjustment variance for the initial adjustment: $\mathrm{Var}_E(X)$; the adjustment variance following the additional adjustment by F: $\mathrm{Var}_D(X)$; the difference between these two quantities, being $\mathrm{RVar}_{F/E}(X)$; the partial resolution $\mathrm{R}_{F/E}(X)$; and finally a term amounting to the resolution for the adjusted version of X given E for an adjustment by $[F/E]$: that is, $\mathrm{R}_{[F/E]}(X/E)$. (See [48][section 13]).

Additionally, analagous results for the entire structure $[B]$ are shown. These consist of the the name of the structure: $[B]$; the uncertainty in $[B]$ unresolved after the initial adjustment: $\mathrm{U}_E(B)$; the uncertainty in $[B]$ unresolved after the full adjustment: $\mathrm{U}_D(B)$; the difference between these two uncertainties, being $\mathrm{RU}_{F/E}(B)$; the partial system resolution $\mathrm{R}_{[F/E]}(B)$; and the resolution for the adjusted version $[B/E]$ for the adjustment by $[F/E]$, being $\mathrm{R}_{F/E}(B/E)$.

Resolution partition

▷ ▷ **Usage**

- BD>**option rp** ↩ *switches on resolution partition output.*
- BD>**option -rp** ↩ *switches off resolution partition output.*
- BD>**show (rp)** ↩ *displays the resolution partition for the current adjustment.*

◁ ◁

For every *element* $X \in B$ the output given by the rp option shows the contribution made by each canonical direction to the resolution of X. That is, suppose that there are g canonical directions $\{Z_i\}$ and that the canonical resolutions are $\{\lambda_i\}$. Then we can write

$$\mathrm{R}_D(X) = \frac{\sum_{i=1}^{g} \lambda_i \mathrm{Cov}(Z_i, X)^2}{\mathrm{Var}(X)}$$

and we take the quantity $\frac{\lambda_i \mathrm{Cov}(Z_i,X)^2}{\mathrm{Var}(X)}$ to be the contribution in the i^{th} direction. The rp control can be used to limit the number of contributory directions. The **rp** operator can be used to access individual contributions.

Element with canonical quantity covariances

▷ ▷ **Usage**

- BD>**option lccd** ↩ *switches on element/canonical quantity covariance output.*
- BD>**option -lccd** ↩ *switches off element/canonical quantity covariance output.*
- BD>**show (lccd)** ↩ *displays the element/canonical quantity covariance output for the current adjustment.*

◁ ◁

The lccd option is used to display covariances between the elements of the collection B being adjusted, and the canonical quantities for the adjustment.

Adjustment summary

▷ ▷ **Usage**

- BD>**option ru** ↩ *switches on adjustment summary output.*
- BD>**option -ru** ↩ *switches off adjustment summary output.*
- BD>**show (ru)** ↩ *displays the adjustment summary for the current adjustment.*

◁ ◁

The ru option is used to display a one-line summary of the current fit. The potential sample size is reported together with the resolved uncertainty for the system, $\mathrm{RU}_D(B)$; the resolution for the system, $\mathrm{R}_D(B)$ (expressed as a percentage); and the initial uncertainty for the system, $\mathrm{U}(B)$. It is one of the options usable with the VARYSIZE: command.

Resolution matrix

▷ ▷ **Usage**

- `BD>`**option rm** ↩ *switches on resolution matrix output.*
- `BD>`**option -rm** ↩ *switches off resolution matrix output.*
- `BD>`**show (rm)** ↩ *displays the resolution matrix for the current adjustment.*

◁ ◁

The rm option is used to display the (generally asymmetric) resolution matrix for the full adjustment. For the full rank case, this is equal to

$$(\mathrm{Var}(B))^{\dagger}\mathrm{Cov}(B, D^T)(\mathrm{Var}(D))^{\dagger}\mathrm{Cov}(D, B^T).$$

Variances for the adjustment

▷ ▷ **Usage**

- `BD>`**option v** ↩ *switches on standard adjustment variance output.*
- `BD>`**option v+** ↩ *switches on extended adjustment variance output.*
- `BD>`**option -v** ↩ *switches off adjustment variance output.*
- `BD>`**show (v)** ↩ *displays standard adjustment variance output for the current adjustment.*
- `BD>`**show (v+)** ↩ *displays extended adjustment variance output for the current adjustment.*

◁ ◁

The v and v+ options display information about the variances for the current adjustment. For each *element* $X \in B$ the standard output displays the name of the *element* and its current adjusted variance, $\mathrm{Var}_D(X)$. The extended output also displays its resolved variance, $\mathrm{RVar}_D(X) = \mathrm{Var}(X) - \mathrm{Var}_D(X)$; and its resolution $\mathrm{R}_D(X)$. In the case of exchangeability, also given is the maximal resolution for X, $\lim_{n\to\infty} \mathrm{R}_D(X)$ (representing the maximum resolution possible with an essentially infinite sample size). (See [31][section 3.3].)

In addition various uncertainties for the overall belief structure $[B]$ are presented. These are the name of the structure being adjusted; its prior, unresolved and resolved uncertainty respectively ($\mathrm{U}(B)$, $\mathrm{U}_D(B)$, and $\mathrm{RU}_D(B)$); and its current system resolution, $\mathrm{R}_D(B)$. In the case of exchangeability, also shown is the system maximal resolution, $\lim_{n\to\infty} \mathrm{R}_D(B)$. (See [31][section 3.7] and [48][section 10.3].)

The v option can be used in both standard and extended modes as one of the options usable with the VARYSIZE: command.

19.4 Accessing the results of observed adjustments and partial adjustments

Consider the following scenario: Suppose that the *base* $\{B\}$ has been adjusted by the *base* $\{D\}$. Occasionally we will be concerned with partial adjustments, and to this purpose for the partial adjustment results we suppose that we adjust initially by the *base* baseE, and then partially by *base* $\{F\}$; and that D represents the complete adjustment $D = E \cup F$. Suppose that we might observe $E = e$, $F = f$, and so too $d = e \cup f$ (otherwise, data-related results - and thus these options - are not available).

Observed adjusted expectations

▷ ▷ **Usage**

- `BD>`**option a** ↩ *switches on standard observed adjusted expectation output.*
- `BD>`**option a+** ↩ *switches on extended observed adjusted expectation output.*
- `BD>`**option -a** ↩ *switches off observed adjusted expectation output.*
- `BD>`**show (a)** ↩ *displays standard observed adjusted expectation output for the current adjustment.*
- `BD>`**show (a+)** ↩ *displays extended observed adjusted expectation output for the current adjustment.*

◁ ◁

The `a` and `a+` options are used for the output of observed adjustments, delivering standard and extended output respectively. The extended output shown depends upon whether or not results are (or will be) available for both an adjustment and a partial adjustment.

Minimal output for an adjustment For a single adjustment the standard output consists of, for every *element* $X \in B$, the name of the *element* and its observed adjusted expectation $\mathrm{E}_d(X)$. (See [31][section 4].)

Extended output for an adjustment For a single adjustment the extended output consists of, for every *element* $X \in B$, the name of the *element*; its initial expectation $\mathrm{E}(X)$; its observed adjusted expectation $\mathrm{E}_d(X)$; and its standardised adjustment $\mathrm{S}_d(X)$. (See [31][section 4.2].)

Extended output for an additional partial adjustment For a partial adjustment the extended output consists of the extended output for a single adjustment listed above, together with the change in observed adjusted expectation:

$$\mathrm{E}_{f/e}(X) = \mathrm{E}_d(X) - \mathrm{E}_e(X),$$

and the standardised value of the change: $\mathrm{S}_{f/e}(X)$. (See [48][sections 6.2.3, 13.2.3].)

Bearing

▷ ▷ **Usage**

- `BD>`**option b** ↩ *switches on bearing output.*
- `BD>`**option -b** ↩ *switches off bearing output.*
- `BD>`**show (b)** ↩ *displays bearing output for the current adjustment.*

◁ ◁

The `b` option is used to display output relating to the size and bearing for the adjustment, and for the partial adjustment if any. (See [31][sections 4.4, 5.4].) Similar output is available using the `b*` option. Both the bearing and the partial bearing may be retained interactively as assignments by using respectively the `b` and `pb` arguments to the KEEP: command. The output shown consists of one or more columns, each representing a linear combination of the *elements* of $\{B\}$:

- firstly, the bearing for the current adjustment, $\mathrm{Z}_d(B)$;
- secondly, in the case of partial adjustment, the bearing for the partial adjustment, $\mathrm{Z}_{f/e}(B)$;
- thirdly, in the case of partial adjustment, the bearing for the initial adjustment, $\mathrm{Z}_e(B)$.

Beneath each column are summary quantities for the observed adjustment: the size of the adjustment, the expected size of the adjustment, and the size ratio respectively. Hence the values reported are (assuming that the partial results are available, otherwise only the first column is reported):

- for the current adjustment, $\mathrm{Size}_d(B)$, $\mathrm{E}(\mathrm{Size}_D(B))$, and $\mathrm{Sr}_d(B)$. The first is available as the operand **size**, and the second is available as the operand **rmtr**.
- for the partial adjustment, $\mathrm{Size}_{f/e}(B)$, $\mathrm{E}(\mathrm{Size}_{F/E}(B))$, and $\mathrm{Sr}_{f/e}(B)$. The first is available as the operand **psize**, and the second is available as the operand **prmtr**.
- for the previous adjustment, $\mathrm{Size}_e(B)$, $\mathrm{E}(\mathrm{Size}_E(B))$, and $\mathrm{Sr}_e(B)$.

Standardised bearing

▷ ▷ **Usage**

- `BD>`**option b*** ↩ *switches on standardised bearing output.*

- `BD>`**option -b*** ↩ *switches off standardised bearing output.*

- `BD>`**show (b*)** ↩ *displays standardised bearing output for the current adjustment.*

◁ ◁

The `b*` option is used to display output relating to the size and standardised bearing for the adjustment, and for the partial adjustment if any. (See [31][sections 4.4, 5.4].) Similar output is available using the `b` option. Both the bearing and the partial bearing may be retained interactively as assignments by using respectively the `b` and `pb` arguments to the KEEP: command, but they cannot be retained in standardised form. The output shown consists of one or more columns, each representing a linear combination of the standardised *elements* of $\{B\}$, where each *element* is standardised to have expectation zero and prior variance unity:

- firstly, the standardised bearing for the current adjustment, $\mathrm{Z}_d(B)$.
- secondly, in the case of partial adjustment, the standardised bearing for the partial adjustment, $\mathrm{Z}_{f/e}(B)$.
- thirdly, in the case of partial adjustment, the standardised bearing for the initial adjustment, $\mathrm{Z}_e(B)$.

Beneath each column are summary quantities for the observed adjustment: the size of the adjustment, the expected size of the adjustment, and the size ratio respectively. Hence the values reported are (assuming that the partial results are available, otherwise only the first column is reported):

- for the current adjustment, $\mathrm{Size}_d(B)$, $\mathrm{E}(\mathrm{Size}_D(B))$, and $\mathrm{Sr}_d(B)$. The first is available as the operand **size**, and the second is available as the operand **rmtr**.
- for the partial adjustment, $\mathrm{Size}_{f/e}(B)$, $\mathrm{E}(\mathrm{Size}_{F/E}(B))$, and $\mathrm{Sr}_{f/e}(B)$. The first is available as the operand **psize**, and the second is available as the operand **prmtr**.
- for the previous adjustment, $\mathrm{Size}_e(B)$, $\mathrm{E}(\mathrm{Size}_E(B))$, and $\mathrm{Sr}_e(B)$.

Correlations between bearings and elements

▷ ▷ **Usage**

- `BD>`**option becorr** ↩ *switches on bearing-element correlation output.*

- BD>**option -becorr** ↩ *switches off bearing-element correlation output.*

- BD>**show (becorr)** ↩ *displays bearing-element correlation output for the current adjustment.*

◁ ◁

The becorr option yields the prior correlation between the bearing for the current adjustment, $\mathrm{Z}_d(B)$, and every *element* $X \in B$. If a partial adjustment has been made, then the prior correlations for the bearings for the previous and partial adjustments, $\mathrm{Z}_e(B)$ and $\mathrm{Z}_{f/e}(B)$, with every *element* $X \in B$ are given.

Correlations between bearings

▷ ▷ **Usage**

- BD>**option bcorr** ↩ *switches on bearing inter-correlation output.*

- BD>**option -bcorr** ↩ *switches off bearing inter-correlation output.*

- BD>**show (bcorr)** ↩ *displays bearing inter-correlation output for the current adjustment.*

◁ ◁

When the results of a partial adjustment are available, the bcorr option yields the prior correlations between the three kinds of bearing: the bearing for the current adjustment, $\mathrm{Z}_d(B)$; the previous adjustment, $\mathrm{Z}_e(B)$; and the partial adjustment, $\mathrm{Z}_{f/e}(B)$. One of these three correlations (depending upon the type of fit) is the path correlation, also available using the pc option. In the case of model fitting, the path correlation is the prior correlation between the bearings for the previous and partial adjustments, and in the case of model reduction, it is the prior correlation between the bearings for the current and partial adjustments.

Discrepancy vector

▷ ▷ **Usage**

- BD>**option dv** ↩ *switches on discrepancy vector output.*

- BD>**option -dv** ↩ *switches off discrepancy vector output.*

- BD>**show (dv)** ↩ *displays discrepancy vector output for the current adjustment.*

◁ ◁

The dv option is used to display output relating to the adjustment discrepancy and discrepancy vector for the adjustment, and for the partial adjustment if any. Similar output is available using the dv* option. Both the discrepancy vector and the partial discrepancy vector may be retained interactively as assignments by using respectively the dv and pdv arguments to the KEEP: command. The output shown consists of one or more columns, each representing a linear combination of the *elements* of $\{B\}$:

- firstly, the discrepancy vector for the current adjustment, $\mathrm{Dis}(d)\mathrm{B}$.
- secondly, in the case of partial adjustment, the discrepancy vector for the partial adjustment, $\mathrm{Dis}(f/e)\mathrm{B}$.
- thirdly, in the case of partial adjustment, the discrepancy vector for the initial adjustment, $\mathrm{Dis}(e)\mathrm{B}$.

Beneath each column are summary quantities for the observed adjustment: the discrepancy for the adjustment, the prior expected size of the discrepancy for the adjustment (the expected size of this discrepancy is equal to the rank of the corresponding resolution matrix), and the discrepancy ratio respectively. This summary is also given with the dv* option. Hence the values reported are (assuming that the partial results are available, otherwise only the first column is reported):

- for the current adjustment, $\mathrm{Dis}_d(B)$, $\mathrm{E}(\mathrm{Dis}_D(B))$, and $\frac{\mathrm{Dis}_d(B)}{\mathrm{E}(\mathrm{Dis}_D(B))}$. The first is available as the operand **dvsize**, and the second is available as the operand **rmrank**.
- for the partial adjustment, $\mathrm{Dis}_{f/e}(B)$, $\mathrm{E}(\mathrm{Dis}_{F/E}(B))$, and $\frac{\mathrm{Dis}_{f/e}(B)}{\mathrm{E}(\mathrm{Dis}_{F/E}(B))}$. The first is available as the operand **pdvsize**, and the second is available as the operand **prmrank**.
- for the previous adjustment, $\mathrm{Dis}_e(B)$, $\mathrm{E}(\mathrm{Dis}_E(B))$, and $\frac{\mathrm{Dis}_e(B)}{\mathrm{E}(\mathrm{Dis}_E(B))}$.

Standardised discrepancy vector

▷ ▷ **Usage**

- BD>**option dv*** ↩ *switches on standardised discrepancy vector output.*
- BD>**option -dv*** ↩ *switches off standardised discrepancy vector output.*
- BD>**show (dv*)** ↩ *displays standardised discrepancy vector output for the current adjustment.*

◁ ◁

The dv* option is used to display output relating to the size and standardised discrepancy vector for the adjustment, and for the partial adjustment if any. Similar output is available using the dv option. Both the discrepancy vector and the partial discrepancy vector may be retained interactively as assignments by using respectively the dv and pdv arguments to the KEEP: command, but they cannot be retained in standardised form. The output shown consists of one or more columns, each representing a linear combination of the standardised *elements* of $\{B\}$, where each *element* is standardised to have expectation zero and prior variance unity:

- firstly, the standardised discrepancy vector for the current adjustment, Dis(d)B.
- secondly, in the case of partial adjustment, the standardised discrepancy vector for the partial adjustment, Dis(f/e)B.
- thirdly, in the case of partial adjustment, the standardised discrepancy vector for the initial adjustment, Dis(e)B.

Beneath each column are summary quantities for the observed adjustment: the discrepancy for the adjustment, the prior expected size of the discrepancy for the adjustment (the expected size of this discrepancy is equal to the rank of the corresponding resolution matrix), and the discrepancy ratio respectively. This summary is also given with the dv option. Hence the values reported are (assuming that the partial results are available, otherwise only the first column is reported):

- for the current adjustment, $\mathrm{Dis}_d(B)$, $\mathrm{E}(\mathrm{Dis}_D(B))$, and $\frac{\mathrm{Dis}_d(B)}{\mathrm{E}(\mathrm{Dis}_D(B))}$. The first is available as the operand **dvsize**, and the second is available as the operand **rmrank**.
- for the partial adjustment, $\mathrm{Dis}_{f/e}(B)$, $\mathrm{E}(\mathrm{Dis}_{F/E}(B))$, and $\frac{\mathrm{Dis}_{f/e}(B)}{\mathrm{E}(\mathrm{Dis}_{F/E}(B))}$. The first is available as the operand **pdvsize**, and the second is available as the operand **prmrank**.
- for the previous adjustment, $\mathrm{Dis}_e(B)$, $\mathrm{E}(\mathrm{Dis}_E(B))$, and $\frac{\mathrm{Dis}_e(B)}{\mathrm{E}(\mathrm{Dis}_E(B))}$.

Correlations between discrepancy vectors and elements

▷ ▷ **Usage**

- `BD>`**option dvecorr** ↩ *switches on discrepancy vector-element correlation output.*
- `BD>`**option -dvecorr** ↩ *switches off discrepancy vector-element correlation output.*
- `BD>`**show (dvecorr)** ↩ *displays discrepancy vector-element correlation output for the current adjustment.*

◁ ◁

The `dvecorr` option yields the prior correlation between the discrepancy vector for the current adjustment, Dis(d)B, and every *element* $X \in B$. If a partial adjustment has been made, then the prior correlations for the discrepancy vectors for the previous and partial adjustments, Dis(e)B and Dis(f/e)B, with every *element* $X \in B$ are given.

Correlations between discrepancy vectors

▷ ▷ **Usage**

- `BD>`**option dvcorr** ↩ *switches on discrepancy vector inter-correlation output.*
- `BD>`**option -dvcorr** ↩ *switches off discrepancy vector inter-correlation output.*
- `BD>`**show (dvcorr)** ↩ *displays discrepancy vector inter-correlation output for the current adjustment.*

◁ ◁

When the results of a partial adjustment are available, the `dvcorr` option yields the prior correlations between the three kinds of discrepancy vector: the discrepancy vector for the current adjustment, Dis(d)B; the previous adjustment, Dis(e)B; and the partial adjustment, Dis(f/e)B.

Observed adjusted expectations for canonical directions

▷ ▷ **Usage**

- `BD>`**option cda** ↩ *switches on standard observed adjusted expectation output for the canonical directions.*
- `BD>`**option cda+** ↩ *switches on extended observed adjusted expectation output for the canonical directions.*
- `BD>`**option -cda** ↩ *switches off observed adjusted expectation output for the canonical directions.*
- `BD>`**show (cda)** ↩ *displays standard observed adjusted expectation output for the current adjustment for the canonical directions.*
- `BD>`**show (cda+)** ↩ *displays extended observed adjusted expectation output for the current adjustment for the canonical directions.*

◁ ◁

The `cda` and `cda+` options are used to display standard and extended results for the observed adjustments of canonical directions. Suppose that C is a canonical direction. Then the standard output consists of its name and its observed adjusted expectation, $\mathrm{E}_d(C)$. The extended output consists additionally of its initial expectation (always $\mathrm{E}(C) = 0$ by convention); and its standardised adjustment $\mathrm{S}_d(C)$. Note that the adjusted expectations for the canonical quantities can be obtained using the `cde` or `cde*` options.

Observed adjusted expectations for maximal canonical directions

▷ ▷ **Usage**

- BD>**option mcda** ↩ *switches on standard observed adjusted expectation output for the maximal canonical directions.*

- BD>**option mcda+** ↩ *switches on extended observed adjusted expectation output for the maximal canonical directions.*

- BD>**option -mcda** ↩ *switches off observed adjusted expectation output for the maximal canonical directions.*

- BD>**show (mcda)** ↩ *displays standard observed adjusted expectation output for the current adjustment for the maximal canonical directions.*

- BD>**show (mcda+)** ↩ *displays extended observed adjusted expectation output for the current adjustment for the maximal canonical directions.*

◁ ◁

The mcda and mcda+ options are used to display standard and extended results for the observed adjustments of maximal canonical directions. Suppose that M is a maximal canonical direction. Then the standard output consists of its name and its observed adjusted expectation, $\mathrm{E}_d(M)$. The extended output consists additionally of its initial expectation (always $\mathrm{E}(M) = 0$ by convention); and its standardised adjustment $\mathrm{S}_d(M)$. The option is ignored when there is no adjustment, and also when lack of exchangeability in D implies that there are no maximal results available.

Adjusted means and standard deviations summary

▷ ▷ **Usage**

- BD>**option msd** ↩ *switches on a summary of the effects of adjustment on individual elements.*

- BD>**option -msd** ↩ *switches off a summary of the effects of adjustment on individual elements.*

- BD>**show (msd)** ↩ *displays a summary of the effects of the current adjustment on individual elements.*

◁ ◁

The msd option summarises the output given by the a+ and v+ options for individual *elements* $X \in \{B\}$. For each such element the output consists of the initial and adjusted expectation; the initial and adjusted standard deviations; and the standardised value of the change in expectation.

Path summary

▷ ▷ **Usage**

- BD>**option pathsum** ↩ *enables path summaries and disables all other adjustment output options.*

- BD>**option -pathsum** ↩ *disables output of path summaries.*

◁ ◁

The pathsum option is rather special because its being set disables all other adjustment output options, and the setting of any other adjustment output option likewise disables the pathsum option. This is because the two sorts of option are incompatible, giving messy output.

When switched on, the option gives a summary of sequential observed adjustments, and in particular of the path correlations between successive adjustments. Suppose that the *base* $\{B\}$ is adjusted initially by the *base* $\{G_1\}$, and then partially by the *base* $\{G_2\}$, and so forth; and suppose that $\{G_{[i]}\}$ is the union of the *bases* $G_1 \cup G_2 \ldots \cup G_i$. Suppose that we observe G_i to be g_i for each i, and that we write $g_{[i]}$ for the observed union $\{G_{[i]}\}$.

Suppose that we have adjusted already by $\{G_{[i-1]}\}$. For the i^{th} stepwise partial adjustment, when the *base* $\{G_i\}$ is added, the information displayed is

- the name of the partially adjusting *base* $\{G_i\}$;
- for every *element* $X \in B$, the new current observed adjusted expectation for X, $\mathrm{E}_{G_{[i]}}(X)$;
- the size of the partial adjustment, $\mathrm{Size}_{g_i/g_{[i-1]}}(B)$;
- the expected size of the partial adjustment, $\mathrm{E}(\mathrm{Size}_{G_i/G_{[i-1]}}(B))$;
- and the i^{th} stepwise path correlation $\mathrm{C}(\mathrm{Z}_{g_{[i-1]}}(B), \mathrm{Z}_{g_i/g_{[i-1]}}(B))$.

(See [31][section 5.6].) The path correlation is undefined for the first such adjustment.

Path correlations

▷ ▷ **Usage**

- `BD>`**option pc** ↩ *switches on path correlation output.*
- `BD>`**option -pc** ↩ *switches off path correlation output.*
- `BD>`**show (pc)** ↩ *displays path correlation output for the current adjustment.*

◁ ◁

When the results of a partial adjustment are available, the pc option yields $\mathrm{C}(\mathrm{Z}_e(B), \mathrm{Z}_{f/e}(B))$, the path correlation. In the case of model fitting, this is the prior correlation between the bearings for the previous and partial adjustments, and in the case of model reduction, this is the prior correlation between the bearings for the current and partial adjustments.

Observed adjusted expectations for partial canonical directions

▷ ▷ **Usage**

- `BD>`**option pcda** ↩ *switches on standard observed adjusted expectation output for the partial canonical directions.*
- `BD>`**option pcda+** ↩ *switches on extended observed adjusted expectation output for the partial canonical directions.*
- `BD>`**option -pcda** ↩ *switches off observed adjusted expectation output for the partial canonical directions.*
- `BD>`**show (pcda)** ↩ *displays standard observed adjusted expectation output for the current adjustment for the partial canonical directions.*
- `BD>`**show (pcda+)** ↩ *displays extended observed adjusted expectation output for the current adjustment for the partial canonical directions.*

◁ ◁

The pcda and pcda+ options are used to display standard and extended results for the observed adjustments of partial canonical directions. Suppose that W is a partial canonical direction. Then the standard output consists of its name and its observed adjusted expectation, $\mathrm{E}_d(W)$. The extended output consists additionally of its initial expectation (always $\mathrm{E}(W) = 0$ by convention); and its standardised adjustment $\mathrm{S}_d(W)$. The option is ignored when there is no partial adjustment.

19.5 Plotting options

Connecting consecutive points

▷ ▷ **Usage**

- BD>**option connect** ↩ *for high-resolution plots, enable the connection of consecutive* (x, y) *points.*

- BD>**option -connect** ↩ *for high resolution plots, disable connection of consecutive* (x, y) *points.*

◁ ◁

The connect option applies only to the production of high resolution plots using the PLOT: command. When this option is enabled, consecutive points are connected. For example, suppose that the plot is of x versus y and z simultaneously. A line will be drawn between (x_1, y_1) and (x_2, y_2), between (x_2, y_2) and (x_3, y_3), and so forth; and also between (x_1, z_1) and (x_2, z_2), between (x_2, z_2) and (x_3, z_3), and so forth;

Removing plotted points

▷ ▷ **Usage**

- BD>**option nopoints** ↩ *for high-resolution plots, disables the plotting of* (x, y) *points.*

- BD>**option -nopoints** ↩ *for high resolution plots, disables the plotting of* (x, y) *points.*

◁ ◁

The nopoints option applies only to the production of high resolution plots using the PLOT: command. It is used in combination with the connect option. When invoked, high resolution plots do not show points – only the lines connecting them.

19.6 Miscellaneous options

All output

▷ ▷ **Usage**

- BD>**option all** ↩ *switches on all standard adjustment-output options except the pathsum option.*

- BD>**option all+** ↩ *switches on all of the standard and extended adjustment output options except the* pathsum *option.*

- BD>**option -all** ↩ *switches off all adjustment-output options.*

- BD>**show (all)** ↩ *displays all standard output for the current adjustment.*

- BD>**show (all+)** ↩ *displays all standard and extended output for the current adjustment.*

◁ ◁

When the all and all+ options are used with the OPTION: command, they are used to enable or disable quickly all adjustment output options. When used with the SHOW: command, they display all available output for the current adjustment. The pathsum option is treated somewhat differently in that it is excluded from the list of options for these purposes.

Belief comparison canonical quantities

▷ ▷ **Usage**

- BD>**option bcd** ↩ *switch on standard belief comparison output.*

- BD>**option bcd+** ↩ *switch on extended belief comparison output.*

- BD>**option -bcd** ↩ *switch off the output of belief comparison directions.*

◁ ◁

The bcd and bcd+ options are used to control the appearance of directions as well as corresponding variance quantities for the comparison of beliefs offered by the COMPARE: command.

Detecting data inconsistencies

▷ ▷ **Usage**

- BD>**option datawarn** ↩ *summarise data inconsistencies, if any.*

- BD>**option datawarn+** ↩ *switch on the output of details of data inconsistencies, if any, together with linear combinations.*

- BD>**option -datawarn** ↩ *switch off the output of details of data inconsistencies.*

◁ ◁

The datawarn and datawarn+ options are used to output details of data inconsistencies, if any, as they arise during adjustments. Suppose that the data quantities D for an adjustment are observed to be d. These data are inconsistent with the beliefs specified beforehand if, for any linear combination $q^T(D - \mathrm{E}_D())$, we have $\mathrm{Var}(q^T(D - \mathrm{E}_D())) = 0$ and $q^T(d - \mathrm{E}_D()) \neq 0$. If the datawarn+ option is switched on, both the values of $q^T(D - \mathrm{E}_D())$ and the linear combination q are reported. Otherwise only the value of the discrepancy from zero is reported. Such discrepancies usually cause errors, but instead a warning can be reported if the datawarn control is used.

Program command lines

▷ ▷ **Usage**

- BD>**option echo** ↩ *switch on the echo commands to the screen.*

- BD>**option -echo** ↩ *switch off the echo of commands to screen.*

◁ ◁

The echo option can used to echo all [B/D] commands to the screen. Its main use is to keep track of the commands being obeyed during a macro. For the WINDOWS version, it also displays the memory available at the commencement of the current command.

Eigenanalyses

▷ ▷ **Usage**

- BD>**option eig** ↩ *enable eigenvalue output.*
- BD>**option eig+** ↩ *enable eigenvalue and eigenvector output.*
- BD>**option -eig** ↩ *disable eigenvalue and eigenvector output.*

◁ ◁

The eig and eig+ options are used to output the eigenstructure determined using an EIGEN: command. The eigenvalues are obtained by using the option eig, and both eigenvalues and eigenvectors are obtained by using the option eig+.

Nonnegative definite matrix corrections

▷ ▷ **Usage**

- BD>**option eigadd** ↩ *enable the display of a matrix correction.*
- BD>**option -eigadd** ↩ *disable the display of a matrix correction.*

◁ ◁

The eigadd option is used in conjunction with the ADJUST:, SCAN:, EIGEN: and INVERT: commands. It has two roles. Firstly, it determines whether diagnostic aid is displayed in the case of the detection of negative eigenvalues. For most purposes we require a matrix C to be non-negative definite. When the eigensystem determination algorithm within [B/D] detects negative eigenvalues beyond a certain threshhold, and when this option is enabled, a matrix M is displayed such that the matrix $C + M$ is non-negative definite.

It is the case that [B/D] needs to make judgements about the ranks of several matrices (assumed non-negative definite). The nature of the routines here is that very small eigenvalues can represent inaccurately determined eigenvalues which should equal zero. Thus, we take any eigenvalue smaller than a given threshhold to be equal to zero. The threshhold used is a small fraction of the euclidian norm of the subject matrix. The second function of the eigadd option is to report the value of the threshhold and display all eigenvalues smaller than it in the case where [B/D] judges the rank of the matrix to be smaller than its dimension.

Influence diagrams

▷ ▷ **Usage**

- BD>**option influence** ↩ *enable interactive output of influence diagrams during adjustments.*
- BD>**option -influence** ↩ *disable interactive output of influence diagrams during adjustments.*

◁ ◁

The influence option permits the interactive construction of influence diagrams during adjustments being performed with the ADJUST: command.

Note that the overwrite control can be used to superimpose successive influence diagrams from a sequence of adjustments, and the overshade control can then be used to control the type of shading resulting from partial adjustments.

Partial correlation diagrams

▹ ▹ **Usage**

- BD>**option pcdiag** ↩ *enable interactive output of partial correlation diagrams.*
- BD>**option -pcdiag** ↩ *disable interactive output of partial correlation diagrams.*

◃ ◃

The `pcdiag` option is used only as a switch to determine whether or not the partial correlation diagram is produced when the PCDIAG: command is issued.

Correlations for data summaries

▹ ▹ **Usage**

- BD>**option scorr** ↩ *enable output of the scorrelation matrix.*
- BD>**option -scorr** ↩ *disable output of the correlation matrix.*

◃ ◃

The `scorr` option is used as a switch to determine whether a correlation matrix is produced for the *data carrier*s supplied as the arguments to a SUMMARY: command. See also the `scov` option, which yields sample covariances.

Covariances for data summaries

▹ ▹ **Usage**

- BD>**option scov** ↩ *enable output of the covariance matrix.*
- BD>**option -scov** ↩ *disable output of the covariance matrix.*

◃ ◃

The `scov` option is used as a switch to determine whether a covariance matrix is produced for the *data carrier*s supplied as the arguments to a SUMMARY: command. See also the `scorr` option, which yields sample correlations.

Warning messages

▹ ▹ **Usage**

- BD>**option warn** ↩ *switches on warning messages.*
- BD>**option -warn** ↩ *switches off warning messages.*

◃ ◃

The `warn` option is used to switch on or off the display of warning messages. Error messages are always displayed.

Chapter 20

Controls

20.1 Setting controls

▷ ▷ **Syntax**

BD>**control:** [-]**Argument1** [,[-]**Argument2**] [, ...] ↩

where Argument1, Argument2, ... *are arguments of the form discussed below.*

◁ ◁

The CONTROL: command is used to set various switches, with a general syntax as given above. The preceding of an argument by a minus sign means that the switch referred to is to be switched off. Whereas the OPTION: command sets switches to handle various output alternatives, the former affects program flow more fundamentally. Many controls are boolean switches, but some others take numeric arguments. A list of possible controls is available by issuing the LOOK: command with argument controls.

20.2 Controls affecting default expectation and belief stores

Change expectation store default

▷ ▷ **Usage**

- BD>**control e=I** ↩ *where I is an integer representing a valid* expectation store *number.*
- *the default is expectation store number $I = 1$.*

◁ ◁

The e control sets the default expectation store. Commands which need to refer to element expectations and take their expectations from the default expectation store are as follows: ELEMENT:, E:, EXPORT:, ADJUST:, ITADJUST:, SCAN:, COMPARE:. Additionally, the **ex** operator can use the default expectation store. The current value of the default can be seen by issuing the LOOK: command with argument nc.

Change belief store default

▷ ▷ **Usage**

- BD>**control v=I** ↩ *where I is an integer representing a valid* belief store *number.*
- *the default is belief store number $I = 1$.*

◁ ◁

The v control sets the default belief store. This is only referenced when the simplified form of the **var** operator is used. The current value of the default can be seen by issuing the LOOK: command with argument nc.

20.3 Controls affecting exchangeable adjustments

Force use of exchangeable routines

▷ ▷ **Usage**

- BD>**control -exchangeable** ↩ *switches off the control.*
- BD>**control exchangeable** ↩ *switches on the control.*
- *default setting is OFF.*

◁ ◁

The exchangeable control can be used to force the exploitation of exchangeability in the special circumstances where (a) the sample size (fictitious or actual) is unity; and (b) beliefs specified for exchangeable situations exist. Ordinarily a sample size of unity doesn't require these beliefs for the evaluation of the adjustment. Use of this control implies that the modelvar control also will be referenced. This method of forcing the exploitation of exchangeability is not available for the iterative adjustment algorithm.

When this method is used, then all the features commonly available to exploit exchangeability are available even though the original sample size was unity.

Set fictitious sample size

▷ ▷ **Usage**

- BD>**control obs=E** ↩ *where E is any valid* equation *which, when rounded, results in a positive integer.*
- *the default value is $E = 1$.*

◁ ◁

The obs control determines the fictitious sample size that the program assumes when it is preparing an adjustment in the case that no actual data is available, or in the case that the usedata control is switched off. If a mistake is made in defining the equation, and an error is reported, then the previous value will be retained.

Use available data

▷ ▷ **Usage**

- BD>**control -usedata** ↩ *switches off the control.*
- BD>**control usedata** ↩ *switches on the control.*
- *default setting is ON.*

◁ ◁

The usedata control determines whether the command uses any available data to produce observed adjustments. When the control is on, the program always tries to use data if available. If no data is available, or if the control was switched off, there is no attempt to use data, and the prospective sample size used is the one set by the obs control.

Using special algorithms for exchngeable adjustments

▷ ▷ **Usage**

- BD>**control -matchcd** ↩ *switches the control off.*
- BD>**control matchcd** ↩ *switches the control on.*
- *the default is OFF.*

◁ ◁

The setting of the matchcd control determines whether or not mean component and predictive component exchangeable adjustments carried out via the ADJUST: command are computed using special algorithms. We recommend that the special algorithms should be used as this avoids much unnecessary calculation. Except in the case of rank degeneracy, the results will be identical. In the case of rank degeneracy, the special algorithms ensure that the canonical directions for mean and predictive component adjustments are identical except for a normalising constant; this typically gives canonical directions for the mean component adjustment which are valid but different (because of non-uniqueness) to the directions computed by the default algorithm.

20.4 Data controls

Data are defined as vectors of differing lengths of data observations on *data carriers*. Each vector consists of a sequence $1, 2, \ldots, N$ of cases, and there are two types of case: (1) a real number, representing an observation; (2) undefined. For example that d might be the *data carrier* with data vector (largest case 6) shown in Figure 20.1.

Figure 20.1: A fictitious data vector.

Case	Value	Type
1	-	undefined
2	15.3	observation
3	12.6	observation
4	-	undefined
5	-	undefined
6	11.2	observation

Construct data for a new element

▷ ▷ **Usage**

- BD>**control -builddata** ↩ *switches off the control.*
- BD>**control builddata** ↩ *switches on the control.*
- *default setting is ON.*

◁ ◁

The builddata control is used to determine whether and how data is treated when an *element* is defined by a BUILD: command or a PRODUCT: command. For the PRODUCT: command there are two possibilities. If the builddata control is switched on, the program will construct data as appropriate and where possible. Otherwise no data is constructed. Under the BUILD: command there are four possibilities, depending upon whether or not the control is switched on; and whether or not the *element* being defined is entirely new, or whether it is intended to overwrite a pre-existent *element* having the same name.

- The element preexists and the builddata control is switched on: an attempt is made to construct data from the definition of the RHS in the BUILD: command. Any previously existent data is overwritten or removed.
- The element preexists and the builddata control is switched off: any data which had been defined for the pre-existent *element* becomes attached to the new definition, and there is no attempt made to construct new data.
- The element is entirely new and the builddata control is switched on: an attempt is made to construct data from the definition of the RHS in the BUILD: command.
- The element is entirely new and the builddata control is switched off: no new data is constructed. However, if a *data carrier* with the same name as the new *element* already exists, then the data involved become attached to the new *element*, and the *data carrier* is removed.

In circumstances where an attempt is made to construct data from the definition part of the BUILD: command, the maximum possible subset of cases for which there exists data is used. See Figure 6.3 for the complete scheme.

Automatic data selection

▷ ▷ **Usage**

- BD>**control -autoselect** ↩ *switches off the control.*
- BD>**control autoselect** ↩ *switches on the control.*
- *default setting is ON.*

◁ ◁

Many of the commands which use data or create it are subject to the autoselect control. When the control is switched off, the *data selection* is whatever has been established by using the SELECT: command, and the total number of selected cases is accessible as the **obs** operand.

When the control is switched on, data are temporarily selected according to the number of observations on a particular key *data carrier*: typically the first *data carrier* encountered during a particular command. For the different commands, the autoselection is defined as follows:

- ADJUST: the first *element* encountered. (If *base* notation is used for *elements* possessing data, *elements* are taken in their order of introduction to [B/D].)
- BUILD: the *data selection* for the newly created *element* is defined to be the subset of observations for which there is a genuine observation on every *data carrier* in the definition part of the command. (DW this is about to change!)
- EXPORT: all possible data
- LOOK: all possible data for the *data-carriers* intended.
- PLOT: and LOPLOT: the first *data carrier* or *element* possessing data encountered. (If *base* notation is used for *element*s possessing data, *element*s are taken in their order of introduction to [B/D].)
- SCAN: the first *element* encountered. (If *base* notation is used for *element*s possessing data, *element*s are taken in their order of introduction to [B/D].)
- SUMMARY: all possible data for the *data-carriers* intended.

Missing value

▷ ▷ **Usage**

- BD>**control missing=I** ↩ *where I is an integer.*
- *the default value is $I = -999$.*

◁ ◁

The missing control is used to set an integer value to represent missing data cases. For example, to use −333 to represent missing observations, issue the command

BD>**control missing = -333** ↩

The control is used in two different ways. Firstly, when exporting data via the EXPORT: command, all completely undefined cases are output as the missing value. Secondly, when data is being entered via a DATA: command into [B/D], this value is used to mark missing values, and the program treats them thereafter as being undefined.

20.5 Storing adjusted covariances and expectations

Storing adjusted covariances

▷ ▷ **Usage**

- BD>**control ac=I** ↩ *where I is zero or an integer representing a valid* belief store *number.*
- *default store number is $I = 0$.*

◁ ◁

Assume the scenario that the *base* $\{B\}$ is to be adjusted by $\{D\}$. The ac control is used to store any evaluated adjusted covariances, $\text{Cov}_D(B, B)$, in a belief store, perhaps for the purpose of using them as further inputs. The default is for them not to be stored, and this default can be restored at any time. Otherwise the adjusted covariances are stored after every adjustment. If the belief store referenced had been locked beforehand, it becomes unlocked (refer to the LOCK: command). The adjusted covariances are also available by using the [B/D] operator **ac**.

Storing scanned adjusted covariances

▷ ▷ **Usage**

- BD>**control scac=I** ↩ *where I is zero or an integer representing a valid* belief store *number.*
- *default store number is $I = 0$.*

◁ ◁

Assume the scenario that the *base* $\{B\}$ has been adjusted by $\{D\}$ ($\{D\}$ might be empty), and that we are about to assess the effect of adjusting additionally by the *base* F by using the SCAN: command. The scac control can only be used in harness with the SCAN: command. Every scan results in potential adjusted covariances, which are generally available via the [B/D] operator **scac**. If this control is used, the potential adjusted covariances $\text{Cov}_{D\cup F}(B, B)$ are retained in the indicated belief store. The default is for them not to be stored, and this default can be restored at any time. Otherwise the potential adjusted covariances are stored after every scan. If the belief store referenced had been locked beforehand, it becomes unlocked (refer to the LOCK: command).

Storing adjusted expectations

▷ ▷ **Usage**

- BD>**control ae=I** ↩ *where I is zero or an integer representing a valid* expectation store *number.*
- *default store number is $I = 0$.*

◁ ◁

Assume the scenario that the *base* $\{B\}$ is to be adjusted by $\{D\}$. The ae control is used to store any evaluated adjusted expectations, $\text{E}_D(B)$, in an expectation store, perhaps for the purpose of using them as further inputs. The default is for them not to be stored, and this default can be restored at any time. Otherwise the adjusted expectations are stored after every adjustment. If the store referenced had been locked beforehand, it becomes unlocked (refer to the ELOCK: command). The adjusted expectations are also available by using the [B/D] operator **aex**.

Storing scanned adjusted expectations

▹ ▹ **Usage**

- BD>**control scae=I** ↩ *where I is zero or an integer representing a valid* belief store *number.*
- *default expectation store is* $I = 0$.

◃ ◃

Assume the scenario that the *base* $\{B\}$ has been adjusted by $\{D\}$ ($\{D\}$ might be empty), and that we are about to assess the effect of adjusting additionally by the *base* F by using the SCAN: command. The scae control can only be used in harness with the SCAN: command. Every scan results in potential adjusted expectations, which are generally available via the [B/D] operator **scae**. If this control is used, the potential adjusted expectations $\mathrm{E}_{D \cup F}(B)$ are retained in the indicated expectation store. The default is for them not to be stored, and this default can be restored at any time. Otherwise the potential adjusted expectations are stored after every scan. If the store referenced had been locked beforehand, it becomes unlocked (refer to the ELOCK: command).

20.6 Belief sourcing and storage for adjustments

All but one of the following controls determine the belief store from which covariances will be taken for all types of adjustment (that is, for the ADJUST:, COHERENCE: and SCAN: commands). The exceptions are the ac and scac controls, used for storing adjusted covariances. Assume the scenario that the *base* $\{B\}$ is to be adjusted by $\{D\}$.

Source for mean component beliefs

▹ ▹ **Usage**

- BD>**control modelvar=I** ↩ *where I is an integer representing a valid* belief store *number.*
- *the default source is belief store number* $I = 1$.

◃ ◃

The modelvar control sets the source for "different situation" variances and covariances specified for the *elements* of $\{D\}$ in the case of exchangeability over $\{D\}$. The setting of this control is referenced for all exchangeable adjustments, and also for coherence checks carried out using the COHERENCE: command.

Source for beliefs for the base to be adjusted

▹ ▹ **Usage**

- BD>**control priorvar=I** ↩ *where I is an integer representing a valid* belief store *number.*
- *the default source is belief store number* $I = 1$.

◃ ◃

The priorvar control sets the source for $\mathrm{Var}(B)$, the variances and covariances specified for the *elements* of $\{B\}$.

Source for covariance beliefs between two bases

▷ ▷ **Usage**

- BD>**control betweenvar=I** ↩ *where I is an integer representing a valid* belief store *number*
- *the default source is belief store number $I = 1$.*

◁ ◁

The betweenvar control sets the source for $\mathrm{Cov}(B, D)$, the covariances specified between the *elements* of $\{B\}$ and the *elements* of $\{D\}$.

Source for mean plus residual component beliefs

▷ ▷ **Usage**

- BD>**control infovar=I** ↩ *where I is an integer representing a valid* belief store *number.*
- *the default source is belief store number $I = 1$.*

◁ ◁

The infovar control sets the source for "same situation" variances and covariances specified for the *elements* of $\{D\}$ in the case of exchangeability over $\{D\}$. Whenever the sample size is unity, this always locates the source for $\mathrm{Var}(D)$.

The setting of this control is referenced for all exchangeable adjustments, and also for coherence checks carried out using the COHERENCE: command.

20.7 Numeric output formatting controls

Setting the number of decimal places

▷ ▷ **Usage**

- BD>**control dplaces=I** ↩ *where I is an integer such that $-1 \leq I \leq 20$.*
- *the default value is $I = 4$.*

◁ ◁

The dplaces control affects the output of certain numbers: The following are possible values for dplaces:

-1 means that a number is to be output in standard notation. The standard is implementation dependent, but will always work somehow;

0 means that the number is to be truncated to its integer part;

1..20 means that the number will be output with the indicated number (1..20) of decimal places.

The setting of the dplaces control currently affects equation values when printed; exported data and exported expectations; and some numbers output during a LOOK: command. The control works in conjunction with the fwidth control, which specifies the minimum width of the number output.

Setting the width of the number output

▷ ▷ **Usage**

- BD>**control fwidth=I** ↩ *where I is an integer such that $-1 \leq I \leq 20$.*
- *the default value is $I = 8$.*

◁ ◁

The fwidth control specifies the minimum total width for the output of certain real numbers. The setting of the control currently affects equation values when printed; exported data and exported expectations; and some numbers output during a LOOK: command. The control works in conjunction with the dplaces control, which specifies the minimum width of the number output.

20.8 Plot controls

Missing value symbol

▷ ▷ **Usage**

- BD>**control missingsymb=C** ↩ *where C is an alphanumeric character or punctuation character.*
- *the default character is the question mark, C =?.*

◁ ◁

The missingsymb control is used to define a missing symbol character. This character will be plotted on a labelled plot whenever there is no alternative.

Number of plot columns

▷ ▷ **Usage**

- BD>**control plotcols=I** ↩ *where I is an integer such that* $30 \leq I \leq 253$.
- *the default value is* $I = 75$.

◁ ◁

The plotcols control is used to set the horizontal resolution for low resolution plots using the LOPLOT: command.

Number of plot lines

▷ ▷ **Usage**

- BD>**control plotlines=I** ↩ *where I is an integer such that* $10 \leq I \leq 250$.
- *the default value is* $I = 18$.

◁ ◁

The plotcols control is used to set the vertical resolution for low resolution plots using the LOPLOT: command.

Multiple y quantities on plot

▷ ▷ **Usage**

- BD>**control -plotxyy** ↩ *switches the control off.*
- BD>**control plotxyy** ↩ *switches the control on.*
- *the default is ON.*

◁ ◁

The plotxyy control is used to determine the type of plot and the order of definition of data carriers to be plotted. When the control is switched on, the plot command takes the first data carrier entered to be the quantity plotted on the x axis, and any remaining entries to be plotted simultaneously on the y axis. When the control is switched off, the plot command takes the first data carrier entered to be the quantity plotted on the y axis, and any remaining entries to be plotted simultaneously on the x axis.

Set x-axis label decimal places

▷ ▷ **Usage**

- BD>**control xaxisdp=E** ↩ *where E is a valid* equation *which should round to an integer in* $[0, 15]$.
- *the default is* $E = 4$.

◁ ◁

The xaxisdp control is used to set the number of decimal places shown for the labels for the x-axis for a high resolution plot. By default, 4 decimal places are output. The default number of decimal places is reset if an error is encountered in specifying E

Set x-axis range

▷ ▷ **Usage**

- BD>**control xscale=E1,E2** ↩ *where $E1$ and $E2$ are valid* equations.
- *the defaults are* $E1 = 0$ *and* $E2 = 0$.

◁ ◁

The xscale control is used to set the range of values for the x-axis for a high resolution plot. By default, the range of values plotted is determined to be the minimum (X_{low}) to the maximum X_{high} of the X quantity to be plotted, and this default may be reset by specifying the values $E1 = 0$ and $E2 = 0$. The default is also set if any error is encountered in specifying either $E1$ or $E2$. The value of $E1$ is ignored if $E1 > X_{low}$, and the value of $E2$ is ignored if $E2 < X_{high}$.

Set y-axis label decimal places

▷ ▷ **Usage**

- BD>**control yaxisdp=E** ↩ *where E is a valid* equation *which should round to an integer in* $[0, 15]$.
- *the default is* $E = 4$.

◁ ◁

The yaxisdp control is used to set the number of decimal places shown for the labels for the y-axis for a high resolution plot. By default, 4 decimal places are output. The default number of decimal places is reset if an error is encountered in specifying E

Set y-axis range

▷ ▷ **Usage**

- BD>**control yscale=E1,E2** ↩ *where $E1$ and $E2$ are valid* equations.
- *the defaults are* $E1 = 0$ *and* $E2 = 0$.

◁ ◁

The yscale control is used to set the range of values for the y-axis for a high resolution plot. By default, the range of values plotted is determined to be the minimum (Y_{low}) to the maximum Y_{high} of the Y quantity to be plotted, and this default may be reset by specifying the values $E1 = 0$ and $E2 = 0$. The default is also set if any error is encountered in specifying either $E1$ or $E2$. The value of $E1$ is ignored if $E1 > Y_{low}$, and the value of $E2$ is ignored if $E2 < Y_{high}$.

20.9 Controls for influence diagrams

Automatic arc drawing

▷ ▷ **Usage**

- `BD>`**control autoarc=I** ↩ *where I is an integer representing an arc type.*
- *the default is $I = 0$.*

◁ ◁

The possible arc types are summarised in Table 11.4. A setting of $I = 0$ means only the arcs defined via grid or arc commands will be drawn. Otherwise, arcs between nodes in the adjustment are automatically drawn in the style intended by the value of I. For example, $I = 3$ gives simple labelled arcs.

A special case occurs where $I = 1$, meaning that only unlabelled directed arcs are to be drawn. For the sake of speed, in this case no scan-type information of the kind discussed in §21.7 is available automatically, because this information is not required for arc labelling. Thus the operands **arcin**, **arcind**, **arcout**, **arcoutd**, and **arcpc** will not be defined and it will be an error to access them. The noscan control can always be used to force the production of this information if necessary.

Set large effect shading size

▷ ▷ **Usage**

- `BD>`**control bigshade=R** ↩ *where $0 < R \leq 1$ is a real number.*
- *the default is $R = 0.5$.*

◁ ◁

The bigshade control is used to set the proportion of area shaded for node and arc size ratios (diagnostic values). Suppose that the size ratio is p and that $p > 1$. Then the area shaded is $1 - (\frac{1}{p})^R$. Thus, the larger the value of R, the larger the area shaded. When $p \leq 1$, the smallshade value is used instead.

Redraw node shading

▷ ▷ **Usage**

- `BD>`**control -overshade** ↩ *switches the control off.*
- `BD>`**control overshade** ↩ *switches the control on.*
- *the default is OFF.*

◁ ◁

The setting of the overshade control is relevant only when influence diagrams are being drawn, and only when the overwrite control is switched on and a previous adjustment exists. In these circumstances, it is possible for node shadings for the influence diagram to change from one adjustment to the next. The overshade control when switched on forces a new node shading to be drawn. Otherwise, the previous node shading is left as it was. This feature will mostly be used to shade nodes in a sequence of adjustments, without losing the shadings at each stage. The overcolour control is used to set the colour for the overshading.

Setting overshade colours

▷ ▷ **Usage**

- BD>**control overcolour=I** ↩ *where* $0 \geq I \leq 15$ *is an integer.*
- *the default is* $I = 4$.

◁ ◁

The overcolour control is used to set the colour used for shading outer node areas representing the overall resolutions for nodes which have been subject to a previous adjustment. This is relevant only when the overshade control is switched on. Table 11.3 contains a list of available colours.

Overwrite graphics screen

▷ ▷ **Usage**

- BD>**control -overwrite** ↩ *switches the control off.*
- BD>**control overwrite** ↩ *switches the control on.*
- *the default is ON.*

◁ ◁

The setting of the overwrite control is relevant only when influence diagrams are being drawn. When set on, the graphics screen is not cleared at the end of the command that produced the screen, and further images drawn on the screen will overwrite what is there already. In this way a complex influence diagram, the result perhaps of a succession of adjustments, can be constructed. The graphics screen may be independently cleared at any time by issuing the GCLEAR: command. When the overwrite control is switched off, the graphics screen is cleared at the end of the command which caused graphics to be drawn.

Omit detailed arc calculations

▷ ▷ **Usage**

- BD>**control -noinfluence** ↩ *switches the control off.*
- BD>**control noinfluence** ↩ *switches the control on.*
- *the default is ON.*

◁ ◁

The setting of the noinfluence control determines whether or not adjustments using the ADJUST: command also scan for the individual effects of adjustment (solitary addition and solitary withdrawal from the overall fit) by each base and element to be used for the adjustment. These are automatically calculated if the `influence` option is switched on, as the information calculated is used to label the arcs on the influence diagram. Otherwise, these calculations can significantly slow down operation of the ADJUST: command. When the control is switched on, the results become available as the operators **arcin**, **arcout**, **arcind**, and **arcoutd**. See also §11.3.

Set small effect shading size

▷ ▷ **Usage**

- BD>**control smallshade=R** ↩ *where* $0 < R \leq 1$ *is a real number.*
- *the default is* $R = 0.5$.

◁ ◁

The smallshade control is used to set the proportion of area shaded for node and arc size ratios (diagnostic values). Suppose that the size ratio is p and that $p \leq 1$. Then the area shaded is $1-p^R$. Thus, the larger the value of R, the larger the area shaded. When $p > 1$, the bigshade value is used instead.

Choose arc label length

▷ ▷ **Usage**

- BD>**control arclength=R** ↩ *where R is a real number such that* $0 \leq R \leq 1$.
- *by default* $R = 0$.

◁ ◁

Arc labels have a length which is generally chosen by default to be an appropriate length to fit between two nodes. The default applies at the outset and otherwise can be reinstated by issuing the arclength control with argument $R = 0$. Otherwise a real value in (0,1) can be supplied, and this forces the arc label to have this length. The entire usable graphics screen is considered to be the unit square.

Choose arc label width

▷ ▷ **Usage**

- BD>**control arcwidth=R** ↩ *where R is a real number such that* $0 \leq R \leq 1$.
- *by default* $R = 0$.

◁ ◁

Arc labels have a width which is generally chosen by default to be appropriate to the situation. The default applies at the outset and otherwise can be reinstated by issuing the arcwidth control with argument $R = 0$. Otherwise a real value in (0,1) can be supplied, and this forces the arc label to have this width. The entire usable graphics screen is considered to be the unit square.

20.10 Controls for canonical wheel diagrams

Show node shadings for canonical quantities

▷ ▷ **Usage**

- BD>**control -wheel** ↩ *switches the control off.*
- BD>**control wheel** ↩ *switches the control on.*
- *the default is ON.*

◁ ◁

The wheel control is used to switch node shadings on influence diagrams from standard to canonical quantities. The control only has any effect when the `influence` option has been selected. See §11.7 for further details.

Set small effect shading size

▷ ▷ **Usage**

- BD>**control spokeshade=R** ↩ *where* $0 < R \leq 1$ *is a real number.*

- *the default is* $R = 8.0$.

◁ ◁

The spokeshade control is used to control the amonut of shading produced by a given canonical quantity diagnostic. Full inner shading corresponds to any size ratio not less than the threshhold of R. Inner shading for spokes with size ratios smaller than R have shading in proportion, so that a size ratio of 2.0 will result in one quarter shading of the inner node when the default of $R = 8.0$ is used. The setting of the spokeshade control is relevant only when the wheel control is switched on. See §11.7 for further details.

Set number of spokes

▷ ▷ **Usage**

- BD>**control spokes=I** ↩ *where* $I \geq 0$ *is an integer.*
- *the default is* $I = 0$*, meaning all.*

◁ ◁

The spokes control is used to set the number of spokes for a canonical wheel. Suppose that an adjustment is carried out for which there are r canonical quantities. Diagnostic shading for node B can therefore be for up to r canonical quantities, with one “spoke” for each canonical quantity. The default value $I = 0$, which means all, will produce node shadings for all r canonical quantities. Alternatively, if you wish to see shadings for the first $q < r$ canonical directions only, issue the control with $I = q$. This will then produce node shadings for $q+1$ spokes, with the extra shading in the final spoke aggregating the diagnostic information for the last $q - r$ canonical directions. This is relevant only when the wheel control is switched on. See §11.7 for further details.

Setting spoke shade colours

▷ ▷ **Usage**

- BD>**control spokecolour=I** ↩ *where* $0 \geq I \leq 15$ *is an integer.*
- *the default is* $I = 10$.

◁ ◁

The spokecolour control is used to set the colour used for shading inner node areas representing diagnostics for canonical quantities for nodes which have been subject to a previous adjustment. This is relevant only when the wheel control is switched on. Table 11.3 contains a list of available colours. See §11.7 for further details.

20.11 Controls for partial correlation results

Setting the belief source

▷ ▷ **Usage**

- BD>**control pcsource=I** ↩ *where* I *is an integer representing a valid* belief store *number.*
- *the default source is belief store number* $I = 1$.

◁ ◁

The pcsource control is used to set the source belief store for the variance matrix to be used in a pcdiag command.

Retaining the partial correlations

▷ ▷ **Usage**

- BD>**control pcdest=I** ↩ *where I is an integer equal to zero or representing a valid* belief store *number.*
- *by default $I = 0$, meaning that the partial correlations are not retained.*

◁ ◁

The pcdest control is used to set the destination belief store for the numerical results (partial correlations and multiple correlations) for the collection subject to the pcdiag command. By default the partial correlations are not retained, and a value of $I = 0$ above always restores this default.

Setting the partial correlation threshhold for arcs

▷ ▷ **Usage**

- BD>**control pcarc=R** ↩ *where R is a a real number such that* $0 \leq R \leq 1$.
- *by default $R = 0$.*

◁ ◁

Arcs are drawn between all the nodes on the partial correlation diagram provided that the magnitude of the partial correlation between two nodes exceeds the threshhold value set by the pcarc control. Whenever the control is set to zero a small tolerance (currently 10^{-10}) is allowed so that a calculated partial correlation of 10^{-11} is taken as zero.

20.12 Controlling the size of the graphics window

Setting the horizontal resolution

▷ ▷ **Usage**

- BD>**control winxsize=I** ↩ *where I is an integer.*
- *by default $I = 640$ (WINDOWS version), $I = 500$ (UNIX version).*

◁ ◁

The winxsize control is used to set the horizontal resolution in pixels for the graphics window for the WINDOWS and UNIX versions. If the default window appears too big or too small, use this control together with the winysize control to change the window size.

Setting the vertical resolution

▷ ▷ **Usage**

- BD>**control winysize=I** ↩ *where I is an integer.*
- *by default $I = 480$ (WINDOWS version), $I = 500$ (UNIX version).*

◁ ◁

The winysize control is used to set the vertical resolution in pixels for the graphics window for the WINDOWS and UNIX versions. If the default window appears too big or too small, use this control together with the winxsize control to change the window size.

20.13 Miscellaneous controls

Setting path and partial correlation label sizes

▷ ▷ **Usage**

- BD>**control pcradius=R** ↩ *where R is a real number such that* $0 \leq R \leq 1$.
- *by default* $R = 0$.

◁ ◁

Arcs on influence diagrams can be labelled by the degree of path correlation, and arcs on partial correlation diagrams can be labelled by the degree of partial correlation. The label consists of a small circle, some of which is shaded to indicate the degree of correlation. The radius of this circle has a default value which can be reinstated by issuing the pcradius control with argument $R = 0$. Otherwise a real value in (0,1) can be supplied. The default value is currently one third the radius of the default node size.

Output a system prompt

▷ ▷ **Usage**

- BD>**control -prompt** ↩ *switches off the prompt.*
- BD>**control prompt** ↩ *switches on the prompt.*
- *default setting is ON.*

◁ ◁

The prompt control determines whether [B/D] prints a prompt when it is expecting input from the keyboard. The prompt changes according to the type of input expected:

BD> is the normal prompt, indicating that [B/D] expects a command which it will then obey;

BD< is as above, but also indicates that the program is within an active loop set by preceding IF: or ELSE: commands;

BD/ indicates that the program is within a loop set by a preceding FOR: command, and any input given can only be acted upon when the end of the loop becomes known;

BD- indicates that the program expects non-command input, such as a string, function or constant declaration;

BD* indicates that the program expects strictly numerical input;

BD& indicates that the program expects the continuation of a command line.

Repeat warning and error messages

▷ ▷ **Usage**

- BD>**control -repeatmessage** ↩ *switches the control off.*
- BD>**control repeatmessage** ↩ *switches the control on.*
- *the default is ON.*

◁ ◁

The repeatmessage control is used to set whether a text message appears together with a message number in the case of an error or warning occurring. When the control is switched on, any message number is always accompanied by a message. When the control is switched off, any message number is always accompanied by a message unless the number repeats the previous message number.

Report data inconsistencies

▷ ▷ **Usage**

- BD>**control -datawarn** ↩ *switches the control off.*
- BD>**control datawarn** ↩ *switches the control on.*
- *the default is OFF.*

◁ ◁

The datawarn control is used to set whether a data inconsistency detected during an adjustment is reported as an error or as a warning message. In the former case, the adjustment is terminated, and details may be reported if the `datawarn` or `datawarn+` option has been switched on. Otherwise, a warning will be reported and the adjustment will continue **at your own risk**!

Set level of resolution partition detail

▷ ▷ **Usage**

- BD>**control rp=I** ↩ *where I is a non-negative integer not larger than the value given by the* **rmrank** *operand.*
- *the default value is $I = 0$.*

◁ ◁

The rp control determines how many contributing directions appear in the resolution partition output made available via the corresponding output option `rp`. The possible values are

0 means that all contributing directions are to be output. The number of contributing directions is equal to the operand **rmrank**, representing the number of non-zero canonical resolutions in an adjustment.

1..rmrank-2 means that only the most important ($1 \ldots$ **rmrank** $- 2$) contributing directions will be output, together with a final column of figures representing the contributions from the least important remaining caonical directions, **rmrank** $- 1 \ldots$ **rmrank**.

Set level of titling

▷ ▷ **Usage**

- BD>**control -titles** ↩ *switches the control off.*
- BD>**control titles** ↩ *switches the control on.*
- *the default is ON.*

◁ ◁

The titles control affects the quantity of titling output. When switched on, full titling accompanies various outputs. Otherwise the output is terser.

20.14 Controls affecting belief comparisons and comparison diagrams

Belief sourcing for the comparison of hypotheses

▷ ▷ **Usage**

- BD>**control compare = I1, I2** ↩ *where I1 and I2 are integers representing valid* belief store *numbers.*
- *The default values are* $I1 = 1$ *and* $I2 = 2$.

◁ ◁

The compare control determines the belief stores from which covariances will be taken for the comparison of alternative variance hypotheses. Future belief comparisons then compare the alternative specifications $\text{Var}_{I1}(B)$ to (over) $\text{Var}_{I2}(B)$ for the *base* $\{B\}$.

Setting the diagram standard deviation scale

▷ ▷ **Usage**

- BD>**control cdscaler = I** ↩ *where I is an integer.*
- *The default value is* $I1 = 3$ *standard deviations.*

◁ ◁

The cdscaler control specifies the number of standard deviations (relative to the model having the largest variance in that quadrant) represented by the node radius. It is used to determine how far along the radius, from centre to boundary, an observed value will be plotted. An observed value of 1.5 will be plotted half way to the boundary, by default. An obseved value of more than three will be plotted on the boundary, and shaded differently.

Setting the radius of the observation mark

▷ ▷ **Usage**

- BD>**control cdradius=R** ↩ *where R is a real number such that* $0 \leq R \leq 1$.
- *by default* $R = 0$.

◁ ◁

Observed values of eigenvectors comparing two alternative specifications are marked as small circles in quadrants on a belief comparison diagram. The label consists of a small circle, whose radius has a default value which can be reinstated by issuing the pcradius control with argument $R = 0$. Otherwise a real value in (0,1) can be supplied. The default value is currently one third the radius of the default node size.

Chapter 21

[B/D] operands and operators

21.1 General remarks

- A legal *equation* is an infix sequence of operators and operands. In addition to the usual operators described in §21.2 and §21.9 a considerable number of operators and operands peculiar to [B/D] are available, allowing access to [B/D] outputs. These are described in the remaining sections of this chapter.
- A pair of round brackets is generally used to delimit an *equation*. In certain situations the delimiting pair of brackets, (...) is redundant, and may be omitted. However, in doubt you should enclose an *equation* within round brackets - they cannot harm.
- A pair of round brackets enclosing nothing, namely (), causes an error.
- Boolean operators have the lowest priority in *equations*. For example, (1=1*2) is evaluated as 0, because (1*2) is evaluated first.
- Functional forms (defined within FE: and FVAR: commands) are allowed within equations. See the relevant [B/D] operators **ex**, **corr**, and **var**.
- A list of operands and operators is available interactively by issuing the LOOK: command with arguments operands and operators respectively.

21.2 Binary operators

The binary operators available are shown in Figure 21.1.

Figure 21.1: Binary operators for real operands x, y.

Operand A	Operator	Operand B	Result
x	$+$	y	$x+y$
x	$-$	y	$x-y$
x	$*$	y	xy
x	/	$y \neq 0$	x/y
x	$\wedge$	y	x^y
x	$=$	y	1 if $x=y$, 0 otherwise
x	<>	y	1 if $x \neq y$, 0 otherwise
x	>	y	1 if $x > y$, 0 otherwise
x	<	y	1 if $x < y$, 0 otherwise
x	<=	y	1 if $x \leq y$, 0 otherwise
x	>=	y	1 if $x \geq y$, 0 otherwise

21.3 Accessing the results of an adjustment

Consider the following scenario: Suppose that *base* $\{B\}$ has been adjusted by *base* $\{D\}$ and that we might observe $D = d$ (otherwise, data-related results are not available).

Adjusted expectation

▷ ▷ **Usage**

- **aex** *operator; one argument* N *in parenthesis.* N *is the name of an* element.

◁ ◁

The value returned is the current evaluation of the adjusted expectation for any newly adjusted *element* N. For example, suppose that *element* N is contained in *base* $\{B\}$. Then $\mathbf{aex}(N) = \mathrm{E}_d(N)$. See [31, sections 3.1, 4].

Adjusted covariance

▷ ▷ **Usage**

- **ac** *operator; two arguments* N_1 *and* N_2 *in parenthesis, and separated by a comma.* N_1 *and* N_2 *are* element*s.*

◁ ◁

The value returned is the current adjusted covariance between any two newly adjusted *elements* N_1 and N_2. For example, suppose that N_1 and N_2 are contained in *base* $\{B\}$. Then

$$\mathbf{ac}(N_1, N_2) = \mathrm{Cov}_D(N_1, N_2).$$

See [31, section 3.3]. See also the **av** operator if only variances are required.

If $\{B\}$ comprises the *elements* $Y = [Y_1, Y_2, \ldots, Y_k]$, and c is some k-dimensional vector, and if we define the $k \times k$ matrix $H_{ij} = \{\mathbf{ac}(Y_i, Y_j)\}$, then the adjusted variance of $c^T Y$ is

$$\mathrm{Var}_D(c^T Y) = c^T H c,$$

and H is the adjusted belief structure. Note also that

$$\begin{aligned} \phi_i^T H \phi_i &= 1 - r_i \\ \phi_i^T H \phi_j &= 0 \end{aligned}$$

where the r_i's are the canonical resolutions for the adjustment (these are obtainable using the [B/D] operator $r_i = \mathbf{cr}(i)$), and where ϕ_i is the corresponding eigenvector (which may be retained using the cd argument to the KEEP: command). The diagonal values $H_{ii} = \mathbf{ac}(Y_i, Y_i)$ are what you see if you issue the command SHOW: (v).

Adjusted variance

▷ ▷ **Usage**

- **av** *operator; one argument* n_1 *in parenthesis.* n_1 *is the name of an* element.

◁ ◁

the value returned is the current adjusted variance for the newly adjusted *element* n_1. for example, suppose that n_1 is contained in *base* $\{b\}$. then $\mathbf{av}(n_1) = \mathrm{Var}_d(n_1)$. see [31, section 3.3]. this operator is shorthand for the **ac** operator with the same two arguments. that is, $\mathbf{av}(n_1)$ is equivalent to $\mathbf{ac}(n_1, n_1)$.

Size of the adjustment

▷ ▷ **Usage**

- **size** *operand.*

◁ ◁

If data is available, the size of an adjustment (the variance of the bearing for the adjustment) is returned, that is $\mathrm{Size}_d(B)$. See [31, section 4.4].

Discrepancy for the adjustment

▷ ▷ **Usage**

- **dvsize** *operand.*

◁ ◁

If data is available, the discrepancy for an adjustment (the largest squared change in expectation, relative to the amount of prior variation removed) is returned, that is $\mathrm{Dis}_d(B)$.

Last kind of adjustment

▷ ▷ **Usage**

- **lasttype** *operand.*

◁ ◁

This returns a signed integer according to the last type of adjustment:

-1 An adjustment involving deletion of *elements* from D, with no data observed.

1 An adjustment involving addition of *elements* to D, with no data observed.

-2 An adjustment involving deletion of *elements* from D, with data observed.

2 An adjustment involving addition of *elements* to D, with data observed.

Thus, the sign of the value given indicates whether the command had involved addition to or deletion from D; and the absolute value of the integer indicates whether observations $D = d$ were available or not.

Rank of prior variance matrix

▷ ▷ **Usage**

- **rank** *operand.*

◁ ◁

This returns $\mathrm{r}(\mathrm{Var}(B))$, the rank of the prior variance matrix for the collection $\{B\}$ being adjusted.

Maximal resolution matrix rank

▷ ▷ **Usage**

- **mrmrank** *operand.*

◁ ◁

This returns $\mathrm{U}_{D,n\to\infty}(B)$: the rank of the maximal resolution matrix, being the number of maximal canonical directions corresponding to non-zero maximal canonical resolutions. This quantity is available only when an adjustment involving exchangeable beliefs has been performed.

Maximal resolution matrix trace

▷ ▷ **Usage**

- **mrmtr** *operand.*

◁ ◁

This returns $\mathrm{RU}_{D,n\to\infty}(B)$, the trace of the maximal resolution matrix, equal to the sum of the maximal canonical resolutions. This quantity is available only when an adjustment involving exchangeable beliefs has been performed.

Kind of adjustment

▷ ▷ **Usage**

- **adjhist** *operand.*

◁ ◁

This returns one of three integers according to the type of adjustment performed:

1 is returned if a single adjustment has occurred.

2 is returned if an initial adjustment followed by a partial adjustment has occurred, and where the partial adjustment is null. In such situations, the partial resolution matrix is null.

3 is returned if an initial adjustment followed by a partial adjustment has occurred, and where the partial adjustment is non-null.

Resolution matrix rank

▷ ▷ **Usage**

- **rmrank** *operand.*

◁ ◁

This returns $\mathrm{U}(B)$: the rank of the resolution matrix, being the number of canonical directions corresponding to non-zero canonical resolutions. See [31, sections 3.6,3.7].

Resolution matrix trace

▷ ▷ **Usage**

- **rmtr** *operand.*

◁ ◁

This returns $\mathrm{RU}_D(B)$, the trace of the resolution matrix, equal to the sum of the canonical resolutions. See [31, sections 3.6, 3.7].

Canonical resolutions

▷ ▷ **Usage**

- **cr** *operator; one argument i in parenthesis. i is any valid* equation*, rounded to the nearest integer.*

◁ ◁

This returns the ith largest canonical resolution. It is an error if the index i is smaller than one or greater than the rank of the resolution matrix (obtainable as the operand **rmrank**). For example, suppose that W_4 is the 4th canonical direction for the adjustment of $\{B\}$ by $\{D\}$. Then **cr**$(4) = \mathrm{R}_D(W_4)$. See [31, section 3.6]

Resolution partition

▷ ▷ **Usage**

- **rp** *operator; two operands, N followed by E, in parenthesis and separated by a comma. N is the name of an* element. *E is any valid* equation. *The result of the equation, when rounded, should be a positive integer i.*

◁ ◁

This returns the contribution to resolution in *element* N offered by the i^{th} canonical direction. (See the rp control for additional information.) It is an error if the index i is smaller than one or greater than the rank of the resolution matrix (obtainable as the operand **rmrank**).

An example of the use of the **rp** operator is as follows, where *Temperature* is assumed to be an element in the collection currently being adjusted, and there are at least two canonical directions.

BD>**print: (rp(Temperature,2))** ↩

Sample size for the current adjustment

▷ ▷ **Usage**

- **obs** *operand.*

◁ ◁

This returns the number of selected data observations (and thus the sample size) for the *elements* contained in $\{D\}$ used in the current adjustment. Otherwise, if there is no data available, it returns the fictional sample size being used for the adjustment. Such fictional sample sizes are set using the obs control.

21.4 Accessing results available when expoiting exchangeability

Required sample size

▷ ▷ **Usage**

- **findsize** *operator; two operands, N followed by E, in parenthesis and separated by a comma. N is the name of an* element *or* base. *E is any valid* equation. *The result of the equation, should be a real number a satisfying* $0 < a < 1$.

◁ ◁

This operator may be used only when an exchangeable adjustment of the type introduced in §16.2 is available. Thus, we assume that some adjustment $[B/C(n)]$ has been performed. The **findsize** operator is used to determine sample sizes to guarantee various degrees of resolution for both the collection as a whole and for its component elements. This operator may be used to duplicate the output shown when the FINDSIZE: command is issued.

In the first instance, N should be the name of the collection adjusted, B. The number returned is the sample size n such that $\mathrm{R}_n(B) \geq a$. If the resolution a is unattainable, a value of -1 is returned.

In the second instance, N should be the name of an element $X \in B$ of the collection adjusted, B. Then the number returned is the sample size n such that $\mathrm{R}_n(X) \geq a$. If the resolution a is unattainable, a value of -1 is returned.

An example of the use of the command is as follows, where *Patients* is assumed to be the collection currently being adjusted.

BD>**print: (findsize(Patients,0.95))** ↩

Resolution for a given sample size

▷ ▷ **Usage**

- **resd** *operator; two operands, N followed by E, in parenthesis and separated by a comma. N is the name of an* element *or* base. *E is any valid* equation. *The result of the equation when rounded should be a positive integer m.*

◁ ◁

This operator may be used only with $m > 1$ when an exchangeable adjustment of the type introduced in §16.2 is available. Otherwise the operator can be used with $m = 1$ to return the current resolution. We assume that some adjustment $[B/C(n)]$ has been performed. The **resd** operator is used to calculate the resolution in N for a sample of size m. It works independently of the current sample size, n.

In the first instance, N should be the name of the collection adjusted, B. The number returned is the resolution in B given by the adjustment by $C(m)$, $\mathrm{R}_{C(m)}(B)$. In the second instance, N should be the name of an element $X \in B$ of the collection adjusted, B. The number returned is the resolution in X given by the adjustment by $C(m)$, $\mathrm{R}_{C(m)}(X)$.

Resolved variance for a given sample size

▷ ▷ **Usage**

- **rvard** *operator; two operands, N followed by E, in parenthesis and separated by a comma. N is the name of an* element *or* base. *E is any valid* equation. *The result of the equation when rounded should be a positive integer m.*

◁ ◁

This operator may be used only with $m > 1$ when an exchangeable adjustment of the type introduced in §16.2 is available. Otherwise the operator can be used with $m = 1$ to return the current resolved variance. We assume that some adjustment $[B/C(n)]$ has been performed. The **rvard** operator is used to calculate the resolved variance in N for a sample of size m. It works independently of the current sample size, n.

In the first instance, N should be the name of the collection adjusted, B. The number returned is the resolved variance in B given by the adjustment by $C(m)$, $\mathrm{RVar}_{C(m)}(B)$. In the second instance, N should be the name of an element $X \in B$ of the collection adjusted, B. The number returned is the resolved variance in X given by the adjustment by $C(m)$, $\mathrm{RVar}_{C(m)}(X)$.

Adjusted variance for a given sample size

▷ ▷ **Usage**

- **vard** *operator; two operands, N followed by E, in parenthesis and separated by a comma. N is the name of an* element *or* base. *E is any valid* equation. *The result of the equation when rounded should be a positive integer m.*

◁ ◁

This operator may be used only with $m > 1$ when an exchangeable adjustment of the type introduced in §16.2 is available. Otherwise the operator can be used with $m = 1$ to return the current adjusted variance. We assume that some adjustment $[B/C(n)]$ has been performed. The **vard** operator is used to calculate the adjusted variance in N for a sample of size m. It works independently of the current sample size, n.

In the first instance, N should be the name of the collection adjusted, B. The number returned is the adjusted variance in B given by the adjustment by $C(m)$, $\mathrm{Var}_{C(m)}(B)$. In the second instance, N should be the name of an element $X \in B$ of the collection adjusted, B. The number returned is the adjusted variance in X given by the adjustment by $C(m)$, $\mathrm{Var}_{C(m)}(X)$.

21.5 Accessing the results of a partial adjustment

Consider the following scenario: Suppose that *base* $\{B\}$ has been adjusted initially by *base* $\{D\}$ and then partially by *base* $\{F\}$. Suppose that we might observe $D = d$ and $F = f$ (otherwise, data-related results are not available).

Size of a partial adjustment

▹ ▹ **Usage**

- **psize** *operand.*

◃ ◃

If data is available, the size of a partial adjustment (the variance of the bearing for the partial adjustment) is returned, that is $\mathrm{Size}_{[f/d]}(B)$. See [31, section 5.4].

Discrepancy for a partial adjustment

▹ ▹ **Usage**

- **dvpsize** *operand.*

◃ ◃

If data is available, the discrepancy for a partial adjustment (the largest squared change in expectation, relative to the amount of prior variation removed by the partial adjustment) is returned, that is $\mathrm{Dis}_{f/d}(B)$.

Partial resolution matrix rank

▹ ▹ **Usage**

- **prmrank** *operand.*

◃ ◃

The rank of the partial resolution matrix, being the number of partial canonical directions corresponding to non-zero partial canonical resolutions. See [31, section 5.3].

Partial resolution matrix trace

▹ ▹ **Usage**

- **prmtr** *operand.*

◃ ◃

The trace of the partial resolution matrix, equal to the sum of the partial canonical resolutions. See [31, section 5.3].

Partial canonical resolutions

▹ ▹ **Usage**

- **pcr** *operator; one argument i in parenthesis. i is any valid* equation*, rounded to the nearest integer.*

◃ ◃

This returns the ith largest partial canonical resolution. It is an error if the index i is smaller than one or greater than the rank of the partial resolution matrix (obtainable as the operand **prmrank**). For example, suppose that W_4 is the 4th partial canonical direction for the adjustment of $\{B\}$ by $\{F\}$ given $\{D\}$. Then $\mathbf{pcr}(4) = \mathrm{R}_{[F/D]}(W_4)$. See [31, section 5.3]

Path correlation

▷ ▷ **Usage**

- **pc** *operand.*

◁ ◁

This returns the current path correlation, $\mathrm{C}(d, [f/d])$, or zero if it is undefined. See [31, section 5.5].

21.6 Accessing the results of a scan

We will discuss the results that we might obtain under the following three circumstances where we could issue the SCAN: command.

1. `BD>`**scan: [B/F]** ↩

2. `BD>`**adjust: [B/D]** ↩
   `BD>`**scan: [+/F]** ↩

3. `BD>`**adjust: [B/D+F]** ↩
   `BD>`**scan: [-/F]** ↩

Scanned adjusted expectations

▷ ▷ **Usage**

- **scae** *operator; one arguments N_1 in parenthesis. N_1 is the name of an* element*s.*

◁ ◁

This returns the potential adjusted expectation for N_1, an *element* contained in the *base* $\{B\}$. The value returned for each scenario is:

1. $\mathrm{E}_F(N_1)$
2. $\mathrm{E}_{D\cup F}(N_1)$
3. $\mathrm{E}_D(N_1)$

These results are defined only following the use of the SCAN: command. Notice that these are overall expectations. Partial adjustment results can be found by subtracting, for example, current adjusted expectations.

Scanned adjusted covariances

▷ ▷ **Usage**

- **scac** *operator; two arguments N_1 and N_2 in parenthesis and separated by a comma. N_1 and N_2 are the names of* element*s.*

◁ ◁

This returns the potential adjusted covariance between N_1 and N_2, where these are *elements* contained in the *base* $\{B\}$. The value returned for each scenario is:

1. $\mathrm{Cov}_F(N_1, N_2)$
2. $\mathrm{Cov}_{D\cup F}(N_1, N_2)$
3. $\mathrm{Cov}_D(N_1, N_2)$

These results are defined only following the use of the SCAN: command. Notice that these are overall covariances. Partial adjustment results can be found by subtracting, for example, current adjusted covariances. If only adjusted variances are required, the **scav** operator can also be used.

Scanned resolution

▹ ▹ **Usage**

- **scurvar** *operand.*

◃ ◃

This returns the potential adjustment uncertainty for the *base* $\{B\}$. The value returned for each scenario is:

1. $\mathrm{U}_F(B)$
2. $\mathrm{U}_{D\cup F}(B)$
3. $\mathrm{U}_D(B)$

These results are defined only following a SCAN:. Notice that these are overall uncertainties. Partial adjustment results can be found by subtracting, for example, current adjusted uncertainties.

Scanned partial resolved uncertainty

▹ ▹ **Usage**

- **scptrace** *operand.*

◃ ◃

This returns the potential partial resolved uncertainty remaining in the collection $\{B\}$. The value returned for each scenario is:

1. $\mathrm{RVar}_F(B)$
2. $\mathrm{RVar}_{D\cup F}(B) - \mathrm{RVar}_D(B)$
3. $\mathrm{RVar}_D(B)$

These results are defined only following use of a SCAN: command.

Scanned adjusted variances

▹ ▹ **Usage**

- **scav** *operator; one argument N in parenthesis. N is the name of an* element.

◃ ◃

This returns the potential adjusted variance for N, where N is an *element* contained in the *base* $\{B\}$. The value returned for each scenario is:

1. $\mathrm{Var}_F(N)$
2. $\mathrm{Var}_{D\cup F}(N)$
3. $\mathrm{Var}_D(N)$

These results are defined only following use of a SCAN: command. Notice that these are overall variances. Partial adjustment results can be found by subtracting, for example, current adjusted variances.

Scanned size of an adjustment

▷ ▷ **Usage**

- **scsize** *operand.*

◁ ◁

Not yet implemented.

Scanned size of a partial adjustment

▷ ▷ **Usage**

- **scpsize** *operand.*

◁ ◁

Not yet implemented.

Scanned path correlation for a partial adjustment

▷ ▷ **Usage**

- **scpc** *operand.*

◁ ◁

Not yet implemented.

21.7 Accessing potentials for adjustments

The following results are available under the following scenario: suppose that *base* $\{B\}$ has been adjusted initially by *base* $\{D\}$ and then partially by $\{F = F_1 + F_2 + \ldots + F_r\}$. We will use the notation

$$F_{(i)} = F_1 \cup \ldots \cup F_{i-1} \cup F_{i+1} \cup \ldots \cup F_r$$

to mean the collection F with F_i excluded. Suppose that we might observe $D = d$ and $F_i = f_i$ (otherwise, data-related results are not available). For the following results we need to have issued the ADJUST: command with either the noinfluence control switched off, or with the `influence` option switched on. For example, we might have issued the commands

BD>**control -noinfluence** ↩
BD>**adjust** [+/**F1+F2**] ↩

where $F1$ and $F2$ are the names of *bases*. Typically, the results we are about to describe are used to label the arcs of influence diagrams. These labels can require some time and effort to calculate, and so we don't normally bother - we can switch off the noinfluence control to force their calculation if we need them. Under these conditions, the following results are available.

Source arc influence

▷ ▷ **Usage**

- **arcin** *operator; one argument F_i in parenthesis. F_i is the name of an* element *or* base.

◁ ◁

Under the conditions described above, **arcin**(F_i) returns

$$\mathrm{R}_{[F_i/D]}(B) = \mathrm{R}_{D\cup F_i}(B) - \mathrm{R}_D(B),$$

the partial resolution in the base B resulting from adding F_i as well as D to the adjustment. This value remains until overwritten, so using this quantity if the above conditions are not satisfied can give unpredictable results. The **arcin** operator thus is a measures of the worth of the *base* F_i as a solitary carrier of information. The corresponding operator **arcind** gives the corresponding size ratio as a diagnostic for this quantity when data is available.

Source arc influence diagnostic

▷ ▷ **Usage**

- **arcind** *operator; one argument* F_i *in parenthesis.* F_i *is the name of an* element *or* base.

◁ ◁

Under the conditions described above, and when data is available, **arcind**(F_i) returns

$$\begin{aligned}\mathrm{Sr}_{[f_i/d]}(B) &= \frac{\mathrm{Size}_{[f_i/d]}(B)}{\mathrm{trace}(\mathrm{T}_{[F_i/D]}(B))},\\ &= \frac{\mathrm{Size}_{[f_i/d]}(B)}{\mathrm{Var}(B)\times \mathrm{R}_{[F_i/D]}(B)}.\end{aligned}$$

This is the observed size of the partial adjustment, divided by its expected size, being the trace of the corresponding partial resolution matrix, resulting from adding F_i as well as D to the adjustment of B. This value remains until overwritten, so using this quantity if the above conditions are not satisfied can give unpredictable results. The corresponding operator **arcin** gives the partial resolution used in the divisor.

Destination arc influence

▷ ▷ **Usage**

- **arcout** *operator; one argument* F_i *in parenthesis.* F_i *is the name of an* element *or* base.

◁ ◁

Under the conditions described above, **arcout**(F_i) returns

$$\mathrm{R}_{[F_i/(D\cup F_{(i)})]}(B) = \mathrm{R}_{D\cup F}(B) - \mathrm{R}_{D\cup F_{(i)}}(B),$$

the partial loss in resolution due to withdrawing F_i from the overall adjustment. This value remains until overwritten, so using this quantity if the above conditions are not satisfied can give unpredictable results. The **arcout** operator thus is a measures of the worth of the *base* F_i in supplying information about B that no other source gives. The corresponding operator **arcoutd** gives a diagnostic for this quantity when data is available.

Destination arc influence diagnostic

▷ ▷ **Usage**

- **arcoutd** *operator; one argument* F_i *in parenthesis.* F_i *is the name of an* element *or* base.

◁ ◁

Under the conditions described above, and when data is available, **arcoutd**(F_i) returns

$$\begin{aligned}\mathrm{Sr}_{[f_i/(d\cup f_{(i)})]}(B) &= \frac{\mathrm{Size}_{[f_i/(d\cup f_{(i)})]}(B)}{\mathrm{trace}(\mathrm{T}_{[F_i/(D\cup F_{(i)})]}(B))},\\ &= \frac{\mathrm{Size}_{[f_i/(d\cup f_{(i)})]}(B)}{\mathrm{Var}(B)\times \mathrm{R}_{[F_i/(D\cup F_{(i)})]}(B)}.\end{aligned}$$

the observed size of the partial loss, divided by its expected size, being the trace of the corresponding partial resolution matrix, due to withdrawing $F_i = f_i$ from the adjustment. This value remains until overwritten, so using this quantity if the above conditions are not satisfied can give unpredictable results. The corresponding operator **arcout** gives the partial loss in resolution used in the divisor.

Arc path correlation

▷ ▷ **Usage**

- **arcpc** *operator; one argument F_i in parenthesis. F_i is the name of an* element *or* base.

◁ ◁

Under the conditions described above, and when data is available, **arcpc**(F_i) returns $\mathrm{C}(f_i, (d \cup f_{(i)}))$, the path correlation between the bearing for the node F_i and the aggregated bearing for the remaining nodes being used for the adjustment. Zero is returned if the path correlation is undefined. See [31, section 5.5]. This measures the observed degree of concordance between the information sources F_i and D. The value remains until overwritten, so using this quantity if the above conditions are not satisfied can give unpredictable results.

21.8 Accessing belief sources used for adjustments

The following operands simply return the current values of the various belief source controls.

- **priorvar** returns the current value of the priorvar control.
- **modelvar** returns the current value of the modelvar control.
- **infovar** returns the current value of the infovar control.
- **betweenvar** returns the current value of the betweenvar control.

21.9 Standard arithmetic operators

All arithmetic operators take one argument E in parenthesis, where E is any valid *equation*.

- **abs** returns the absolute value of the argument. For example, **abs**$(1.2 - 3.6) = 2.4$.
- **arctan** is the hyperbolic tangent function, returning the value whose tangent is the given argument. For example, **arctan**$(1) = 0.7854$.
- **cos** returns the cosine of its argument. For example, **cos**$(3.1416) = -1$
- **exp** returns E to the power of its argument. For example, **exp**$(1) = 2.7183$
- **ln** returns the natural logarithm of the argument E. A non-positive argument causes an error. For example, **ln**$(2.7183) = 1$
- **sgn** returns one of three values according to the sign of its argument: zero if the argument is zero; -1 if the argument is negative; and $+1$ if the argument is positive. For example $sgn(-8) = -1$.
- **sin** returns the sine of its argument. For example, **sin**$(3.1416) = 0$.
- **sqr** returns the square of its argument. For example, **sqr**$(1.2) = 1.44$.
- **sqrt** returns the square-root of its argument. An error will be reported if the argument is negative. For example, **sqrt**$(1.44) = 1.2$.
- **trunc** returns the integer truncation of its argument. For example, **trunc**$(1.6) = 1$.

21.10 Data functions

The data operators described in this section all work independently of the current data selection. Thus, each data operator takes into account the maximum possible number of data cases.

Individual data cases can be accessed in equations by giving the name of the data carrier together with the case number in parenthesis - this is the same format as when the data is being defined. For example, we could have

BD>**data: Patientage(3)=18** ↩
BD>**print: (Patientage(3))** ↩

The argument supplied in parenthesis for the observation number may be any valid *equation*, which will be rounded to what should be a positive integer. A simple example is as follows.

BD>**print: (Patientage(1+2))** ↩

It will be an error if the "observation number" thus supplied does not correspond to an observation. The other functions described in this section generally deal with all of the data defined over particular data carriers, rather than individual cases.

Data percentiles
▹ ▹ **Usage**

- **centile** *operator; two operands, D followed by E, in parenthesis and separated by a comma. D is a* data carrier*. E is any valid* equation*. The result of the equation, when rounded, should be an integer I in (0,100).*

◃ ◃

The **centile** operator gives the Ith percentile of currently selected data values on D. For example, if we order the data values as $d_1 \leq d_2 \leq \ldots \leq d_n$, then **centile**$(D, I)$ is the smallest value such that at least I% of the values are less than or equal to this value, and at least $(100 - I)$% of the values are greater than or equal to this value. The lower of two candidates is returned. Note that placing $I = 0$ gives the minimum, and $I = 100$ gives the maximum. All possible data cases defined for the data carrier D are taken into account.

An example of the use of the command is as follows. Suppose that $Temperature$ is the name of a data carrier, and that we require the median. (We generate the 50 artificially to illustrate the use of equations here).

BD>**print: (centile(temperature,25+25))** ↩

Sample correlations
▹ ▹ **Usage**

- **scorr** *operator; two arguments D_1 and D_2 in parenthesis and separated by a comma. D_1 and D_2 are* data carriers.

◃ ◃

This returns the sample correlation between D_1 and D_2. All possible data cases defined both for the data carrier D_1 and the data carrier D_2 are taken into account.

Sample covariances
▹ ▹ **Usage**

- **scov** *operator; two arguments D_1 and D_2 in parenthesis and separated by a comma. D_1 and D_2 are* data carriers.

◃ ◃

This returns the sample covariance between D_1 and D_2 as $\frac{1}{n}$ times the sum of cross product deviations. All possible data cases defined both for the data carrier D_1 and the data carrier D_2 are taken into account.

Highest data case

▷ ▷ **Usage**

- **maxcase** *operator; one operand D in parenthesis. D is a* data carrier.

◁ ◁

This returns the largest case number for which there is data on the *data carrier* D. All possible data cases defined for the data carrier D are taken into account.

Number of matching data cases

▷ ▷ **Usage**

- **match** *operator; two arguments D_1 and D_2 in parenthesis and separated by a comma. D_1 and D_2 are* data carriers.

◁ ◁

This returns the number of cases for which there are observations on both D_1 and D_2. All possible data cases defined both for the data carrier D_1 and the data carrier D_2 are taken into account. The **match** operator is the bivariate analogue of the **number** operator.

Sample mean

▷ ▷ **Usage**

- **mean** *operator; one operand D in parenthesis. D is a* data carrier.

◁ ◁

This returns the arithmetic average of the observations defined for D. All possible data cases defined for the data carrier D are taken into account. If the total number of cases is zero, or if D has not been defined, then the operator returns a value of zero and an error is reported.

Number of data cases

▷ ▷ **Usage**

- **number** *operator; one operand D in parenthesis. D is a* data carrier.

◁ ◁

returns the total number of observations for the *data carrier* D. All possible data cases defined for the data carrier D are taken into account. The **number** operator is the univariate analogue of the **match** operator.

Sample variances

▷ ▷ **Usage**

- **svar** *operator; one operand D in parenthesis. D is a* data carrier.

◁ ◁

This returns the sample variance of D, as $\frac{1}{n}$ times the sum of squared deviations. All possible data cases defined for the data carrier D are taken into account.

21.11 String-handling functions

Detecting null strings

▹ ▹ **Usage**

- **empty$** *operator; one argument S in parenthesis. S is the name of a* string.

◃ ◃

This is a boolean operator which returns the value unity if S is an empty *string* (or if S is not a *string*), and zero if it is a *string* with positive length.

Comparing strings for equality

▹ ▹ **Usage**

- **eq$** *operator; two arguments S_1 and S_2 in parenthesis and separated by a comma. S_1 and S_2 are the names of* strings *or sequences of alphanumeric characters.*

◃ ◃

By default, $S1$ and $S2$ are assumed to be the names of defined *strings*. If either string is not recognised, then the name supplied given is used instead. This is a boolean operator which returns the value unity if the *string* S_1 is equal to the *string* S_2, and zero otherwise. The complementary operator is **neq$**. For example, suppose that we have defined only the strings $a = mustard$ and $b = cress$. Then:

- **eq**(a, b) is equal to zero, as the strings 'mustard' and 'cress' are not equal;
- **eq**$(a, mustard)$ is equal to one, as a is taken to be the string 'mustard' and 'mustard' is not a string, and therefore taken to mean itself.

String length

▹ ▹ **Usage**

- **len$** *operator; one argument S in parenthesis. S is the name of a* string *or a sequence of alphanumeric characters.*

◃ ◃

This returns the length of the *string* S as a non-negative integer. If the *string* has not been defined, then the length of the alphanumeric sequence supplied as the name of the string is returned.

If the string has been defined, but is empty, the length is defined to be zero.

Comparing strings for inequality

▹ ▹ **Usage**

- **neq$** *operator; two arguments S_1 and S_2 in parenthesis and separated by a comma. S_1 and S_2 are the names of* strings *or sequences of alphanumeric characters.*

◃ ◃

By default, $S1$ and $S2$ are assumed to be the names of defined *strings*. If either string is not recognised, then the name supplied given is used instead. This is a boolean operator which returns the value zero if the *string* S_1 is equal to the *string* S_2, and unity otherwise. The complementary operator is **eq$**. For example, suppose that we have defined only the strings $a = mustard$ and $b = cress$. Then:

- **neq**(a, b) is equal to one, as the strings 'mustard' and 'cress' are not equal;

- **neq$**$(a, mustard)$ is equal to zero, as a is taken to be the string 'mustard' and 'mustard' is not a string, and therefore taken to mean itself. Consequently these two are equal, and the operator returns a value indicating false.

Detecting numeric strings

▷ ▷ **Usage**

- **real$** *operator; one argument S in parenthesis. S is the name of a* string *or a sequence of alphanumeric characters.*

◁ ◁

If the name S supplied has not been defined as a string, then the name itself is used as the string. This is an operator which returns the integer value zero, one or two depending upon the form of the *string* S:

1 is returned if S is of the form $d.d$, optionally preceded by a minus sign;

2 is returned if S is of the form d, optionally preceded by a minus sign;

0 is returned if neither of the above applies.

where d represents at least one digit. Note in particular that the decimal point must be both preceded and succeeded by at least one digit if this operator is to return a value of unity. For example, **real$**$(.4) = 0$ and **real$**$(2.1) = 1$.

21.12 Accessing eigenvalue results

The following operators and operands are used to access the results of the INVERT: and EIGEN: commands. Suppose that the $n \times n$ matrix A (A must be symmetric) is supplied to either of these two commands, and suppose that the ordered eigenvalues of A are $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ corresponding to eigenvectors $v_1, v_2, \ldots, v_n$. These eigenvectors are optionally retained as *assignments* using the **eig** argument to the KEEP: command, and might be accessed using the **ascf** operator.

Matrix rank

▷ ▷ **Usage**

- **eigrank** *operand.*

◁ ◁

This returns the number of positive eigenvalues of the matrix A. If the matrix was non-negative definite then this is equivalent to the rank.

Matrix trace

▷ ▷ **Usage**

- **eigtr** *operand.*

◁ ◁

This returns the trace of the matrix A.

Matrix eigenvalues

▷ ▷ **Usage**

- **eig** *operator; one argument i in parenthesis. i is any valid* equation*, rounded to the nearest integer.*

◁ ◁

This returns the ith ordered eigenvalue of the matrix A, namely λ_i, provided that $1 \leq i \leq n$. Otherwise an error occurs.

21.13 Accessing belief inputs

Element expectations

▷ ▷ **Usage**

- **ex** *operator; one argument N in parenthesis. N is the name of an* element*.*
- **ex** *operator; two arguments S, N in parenthesis. S is a expectation store number, N is the name of an* element*.*

◁ ◁

The first form of the usage returns the expectation of the *element* N, with the expectation taken from the current default expectation store. The second form of the usage returns the expectation for the element N with the expectation taken explicitly from expectation store S.

If N is not a built *element*, then the list of functionally specified expectations is searched to determine (and return if present) whether an expectation has been specified functionally via the FE: command.

Notice that you must be careful when using the first form in defining a function, as the default expectation store may be changed (by you) between definition of the function and evaluation of it. The default expectation store here means the default expectation store at the time of evaluation; this might be different to the default at the time of definition. We recommend that you only use the second form when defining functions.

Correlations between elements

▷ ▷ **Usage**

- **corr** *operator; two or three arguments I, N_1, and optionally N_2 in parenthesis and separated by commas. I is an integer representing a valid* belief store*, and N_1 and N_2 are the names of* element*s.*

◁ ◁

This is used to return a belief input from *belief store* I in correlation form. For example, **corr**$(1, N_1, N_2)$ returns the correlation between the *elements* N_1 and N_2 in *belief store* 1. The abbreviated form (allowed only to preserve compatibility with the related operator **var**) with only one argument will always return a value of unity.

Correlations declared implicitly via the FVAR: command may also be accessed using this operator: the built store of names is searched firstly, and the virtual store secondly. For such usage, use of the correlation form assumes that you have made all necessary definitions. For example, use of **corr**$(2, x.1, y.2)$ implies that all the necessary quantities [x with x], [y with y] and [x with y] have been specified beforehand using the FVAR: command. An error will occur otherwise. Note that for both operators **var** and **corr**, the built store of names is searched firstly, and the virtual store secondly. Note also that the beliefs involved are not checked for coherence. Under these circumstances, reported correlations greater than unity are feasible.

Variances and covariances between elements

▷ ▷ **Usage**

- **var** *operator; two or three arguments I, N_1, and optionally N_2 in parenthesis and separated by commas. I is an integer representing a valid* belief store*, and N_1 and N_2 are the names of* element*s.*
- **var** *operator; one or two arguments N_1, and optionally N_2 in parenthesis and separated by commas. N_1 and N_2 are the names of* element*s.*

◁ ◁

The first form of the usage is used to return a belief input from *belief store* I in covariance form. For example, **var**$(1, N_1, N_2)$ returns the covariance specified between the *elements* N_1 and N_2 in *belief store* 1, $\mathrm{Cov}_1(N_1, N_2)$. The abbreviated form with only one argument returns a variance. For example, **var**$(2, N_1)$ is the abbreviation of **var**$(2, N_1, N_1)$ and returns the variance specified for N_1 in *belief store* 2, $\mathrm{Var}_2(N_1)$.

Covariances declared functionally via the FVAR: command may also be accessed using this operator: the built store of names is searched firstly, and the functionally defined store secondly.

The second form of the usage does not require explicitly a belief store number. Instead, the default variance store number is used. This store number is 1 by default, but can be changed by using the v argument to the CONTROL: command. Notice that you must be careful when using the second form in defining a function, as the default variance store may be changed (by you) between definition of the function and evaluation of it. The default variance store here means the default variance store at the time of evaluation; this might be different to the default at the time of definition. We recommend that you only use the first form when defining functions.

21.14 Random number generators

There are two random number generators, providing uniformly and standard normally distributed pseudo-random numbers. Note that these operands have special properties in data assignments. In particular, they are evaluated only as necessary.

Normally distributed random numbers

▷ ▷ **Usage**

- **nrand** *operand.*

◁ ◁

This returns a standardised normal $N(0, 1)$ pseudo-random number with seed set by default or by a SEED: command. The **nrand** operand can act like an operator in some circumstances, for example when used in conjunction with the DATAVEC: command.

Uniformly distributed random numbers

▷ ▷ **Usage**

- **urand** *operand.*

◁ ◁

This returns a uniform $U[0, 1)$ pseudo-random number with seed set by default or by a SEED: command. The **urand** operand can act like an operator in some circumstances, for example when used in conjunction with the DATAVEC: command.

21.15 Other operators and operands

Belief comparison variance summaries

▷ ▷ **Usage**

- **bcd1** *operator; one argument i in parenthesis. i is any valid* equation*, rounded to the nearest integer.*
- **bcd2** *operator; one argument i in parenthesis. i is any valid* equation*, rounded to the nearest integer.*

◁ ◁

The operator **bcd1** returns the "eigenvalue" corresponding to the numerator variance matrix for the i^{th} canonical direction for a belief comparison, and the operator **bcd2** returns similarly the "eigenvalue" corresponding to the denominator variance matrix. It is an error if the index i is smaller than one or greater than the dimension of the variance matrices subjected to the comparison.

Coefficients of components in assignments

▷ ▷ **Usage**

- **ascf** *operator; two arguments* N_1 *and* N_2 *in parenthesis and separated by a comma.* N_1 *is the name of an* assignment, *and* N_2 *the name of an* element, component, *or* assignment.

◁ ◁

This operator returns the coefficient of the component N_2 in the *assignment* N_1. For example, suppose that you have assigned $x = (3)y - (2)z$. Then **ascf**$(x, y) = 3$. The operator takes into account any indices and evaluates functional parts accordingly. In cases where the second argument N_2 is represented more than once in an assignment, the coefficients are accumulated. Where the argument N_2 is not part of the assignment, a coefficient of zero is returned. The following examples may make clear the use of this operator.

BD>**assign: fg.i.j=(11-5+.i+.j)+(4*.j+.i)b.1+(.j)c.12.12+(15)+(.i)+(2+.j)b.i+(.i-2)c.j.i** ↩

BD>**print: (ascf(fg.1.7,b.1)) - should be 38** ↩

BD>**print: (ascf(fg.2.7,b.2)) - should be 9** ↩

BD>**print: (ascf(fg.12.12,c.12.12)) - should be 22** ↩

BD>**print: (ascf(fg.1.124,c.124.1)) - should be -1** ↩

Scalar coefficients in assignments

▷ ▷ **Usage**

- **ascl** *operator; one argument* N *in parenthesis.* N *is the name of an* assignment.

◁ ◁

This returns the scalar part (or zero if there is no scalar part) of the *assignment* whose name is supplied as the argument. For example, suppose that you have assigned $x = (56) + (4)y$ and $t = (15)s$. Then **ascl**$(x) = 56$ and **ascl**$(t) = 0$. The operator takes into account any indices and evaluates functional parts accordingly, as in the following example.

BD>**assign: fg.i.j=(11-5+.i+.j)+(4*.j+.i)b.1+(.j)c.12.12+(15)+(.i)+(2+.j)b.i+(.i-2)c.j.i** ↩

BD>**print: (ascl(fg.2.3)) - should be 28** ↩

Existence of a base

▷ ▷ **Usage**

- **bn** *operator; one argument* N *in parenthesis.* N *is any valid* name.

◁ ◁

This is a boolean operator which returns the value unity if the argument N is the name of a *base*, and zero otherwise.

Elements in a base

▷ ▷ **Usage**

- **count** *operator; one argument* B *in parenthesis.* B *is the name of a* base.

◁ ◁

This returns the number of *elements*, but not *bases* or *data-elements* in the *base* B. If an error occurs whilst the **count** operator is being processed, the number returned is zero.

Elements and bases in a base

▷ ▷ **Usage**

- **countb** *operator; one argument B in parenthesis. B is the name of a* base.

◁ ◁

This returns the number of *elements* or *bases*, but not *data-elements* in the *base B*. If an error occurs whilst the **countb** operator is being processed, the number returned is zero.

Data carriers in a base

▷ ▷ **Usage**

- **countd** *operator; one argument B in parenthesis. B is the name of a* base.

◁ ◁

This returns the number of *data-elements*, in the *base B*. If an error occurs whilst the **countd** operator is being processed, the number returned is zero.

All consituents of a base

▷ ▷ **Usage**

- **countany** *operator; one argument B in parenthesis. B is the name of a* base.

◁ ◁

This returns the total number of constituents of the *base B*, including *data-elements*, other *bases* and *elements*. If an error occurs whilst the **countd** operator is being processed, the number returned is zero.

Data case existence

▷ ▷ **Usage**

- **datain** *operator; two arguments N and I in parenthesis and separated by a comma. N is any valid* name*, and I is an integer.*

◁ ◁

This is a boolean operator which returns the value unity if N is the name of a *data-carrier and* the data case I has been defined for this *data-carrier*, and zero otherwise.

Number of data locations free

▷ ▷ **Usage**

- **dfree** *operand.*

◁ ◁

This returns the number of unused data locations (this calculation may take some time, as it enforces a garbage collection).

Data carrier existence

▷ ▷ **Usage**

- **dn** *operator; one argument N in parenthesis. N is any valid* name.

◁ ◁

This is a boolean operator which returns the value unity if the argument N is the name of a *data carrier* and zero otherwise.

Data cases selected

▷ ▷ **Usage**

- **selection** *operand.*

◁ ◁

This returns the number of data observations selected according to a SELECT: command, irrespective of the setting of the autoselect control.

Number of element locations free

▷ ▷ **Usage**

- **efree** *operand.*

◁ ◁

This returns the number of unused *element* locations (giving the number of *elements* that you can still create).

Element existence

▷ ▷ **Usage**

- **en** *operator; one argument N in parenthesis. N is any valid* name.

◁ ◁

This is a boolean operator which takes the value unity if the argument N is the name of an *element*, and zero otherwise.

Last error number

▷ ▷ **Usage**

- **lasterror** *operand.*

◁ ◁

This returns the last error number, or zero if there have been no errors.

Maximum of two arguments

▷ ▷ **Usage**

- **max** *operator; two arguments E_1 and E_2, each in parenthesis. E_1 and E_2 are any valid* equations.

◁ ◁

This returns the value of whichever of the two arguments is the larger. The complement to this operator is the **min** operator. An example of the use of the command is as follows, where we assume *%theta* to be the name of some constant.

BD>**print: (max(23+7)(30+cos(%theta)))** ↩

Input channel number

▷ ▷ **Usage**

- **mchannel** *operand.*

◁ ◁

This returns an integer in the range $(1, 6)$ representing the current macro channel.

Minimum of two arguments

▷ ▷ **Usage**

- **min** *operator; two arguments* E_1 *and* E_2*, each in parenthesis.* E_1 *and* E_2 *are any valid* equations.

◁ ◁

This returns the value of whichever of the two arguments is the smaller. The complement to this operator is the **max** operator. An example of the use of the command is as follows, where we assume *%theta* to be the name of some constant.

`BD>`**print: (min(23+7)(30+sin(%theta)))** ↩

Area under a standard Normal curve

▷ ▷ **Usage**

- **normal** *operator; one argument E in parenthesis, where E is any valid* equation.

◁ ◁

This returns the area under a standardised Normal $N(0,1)$ curve to the left of the argument E, For example, **normal**$(1.96) = 0.975$. The approximation is accurate to about 10^{-8}.

Value of π

▷ ▷ **Usage**

- **pi** *operand.*

◁ ◁

This returns the value of π correct to 11 decimal places.

Output channel number

▷ ▷ **Usage**

- **ochannel** *operand.*

◁ ◁

This returns an integer in the range $(1,4)$ representing the current output channel.

Computer platform

▷ ▷ **Usage**

- **platform** *operand.*

◁ ◁

This can be used to test for applications for a specific platform. The possible values that could be returned at present, with their implications, are as follows:

1 The [B/D] program has been compiled to run as a UNIX application using the Sun-Berkeley Pascal 2.1 Compiler.

2 The [B/D] program has been compiled to run as a Linux application using the GPP compiler.

3 The [B/D] program has been compiled to run as a WINDOWS application using Borland Pascal With Objects.

Number of belief stores

▹ ▹ **Usage**

- **<u>stores</u>** *operand.*

◃ ◃

This returns the current number of belief stores as an integer. This operand might be used with the <u>LOCK:</u> command, as in the following example which locks all but the first two belief stores:

`BD>`**for: i=3,1,stores | lock: [i]** ↩

Time counter

▹ ▹ **Usage**

- **<u>time</u>** *operand.*

◃ ◃

This returns the time in seconds since the start of the program. The timing is accurate to the nearest second for the UNIX implementation, and to the nearest hundredth of a second for the WINDOWS implementation.

Appendix A

Glossary

% Is the prefix associated with *constants*.

Is the prefix associated with *functions*.

address An address consists of one of the following.

1. @label(c)
2. @label
3. (c)

where "label" is a label, and c some valid channel number, depending upon context. If the channel number (c) is missing, then all current input files are searched for the label. If the label part is missing, the beginning of the file linked to the channel number is assumed.

assignment An assignment is a linear combination of elements, components, and scalars. Each assignment is known by an identifier.

base Bases are the fundamental organisations of elements into collections. In [B/D] they are identified by a sequence of alphanumeric characters, beginning with an alphabetic character. For most applications, single elements may be thought of as bases containing only that element.

belief store Every *element* has variances and covariances associated with it. There are a number (usually at least two) of alternative storage areas for these variances and covariances, corresponding to alternative variance-covariance specifications. Each such area is called a belief store. Belief store number one corresponds to the first belief store, and so forth. The number of belief stores may be found by issuing the LOOK: command using the program argument. See also *e*xpectation store.

case [B/D] ignores the case of all characters entered unless the characters constitute text to be used for titling, or to be printed out directly.

command history A record of the most recent commands issued from the keyboard is retained in the command history. These commands can be replayed using the !: command, or deleted from the buffer using the /: command. The command history can also be saved to a file using the KEEP: command.

command line A command line is generally a [B/D] command, followed by apparopriate arguments to the command. The arguments must be separated from the command by at least one space, or by one colon. The command itself must consist of contiguous characters. Thereafter, as many spaces as desired can be strewn amongst the characters forming the arguments.

component Components are notional elements. They carry no data and have beliefs specified for them only functionally. They take no part in adjustments, but are useful as intermediaries in generating true elements.

constant A constant is a scalar defined by the C: command. Its value is determined at the point where the C: command is used, and remains unchanged until redefined.

current adjustment The current overall adjustment of one collection by all the fitted collections is termed the current adjustment. The current adjustment is undefined if there is no current fit. If there is a current adjustment, there may also be what we term a previous adjustment, and in this case there will be available results concerning the difference between the two.

data-carrier There are two kinds of data carrier: data-elements (which are not associated with any other kind of belief specification), and elements (which are).

data-element Data carriers have a status similar to elements, except that there are no beliefs of any kind associated with them, and they always carry some observations or scalars.

definition part Given a particular command, we call the part to the left hand side of the equals symbol the *naming* part, and the part to the right hand side the *definition* part of the construction. Thus here $Y.i.j$ is the naming part, and $m + a.i + b.j + e.i.j$ is the definition part of the construction.

element Elements are the fundamental belief-carrying quantities, corresponding to random variables in traditional statistical analyses. In [B/D] they are identified by a sequence of alphanumeric characters, beginning with an alphabetic character. One or more period symbols might be used to associate indices with the element. Elements typically are associated with expectations, variances, covariances with other elements and so forth. Additionally, one or more observations may become available on some elements.

equation A legal *equation* is an infix sequence of operators and operands, such as $(2+3/4)$. In addition to the usual operators described in §21.9 and §21.2 a considerable number of operators and operands peculiar to [B/D] are available, allowing access to [B/D] outputs. These are described in detail in the sections of chapter 21. A pair of round brackets is generally used to delimit an *equation*. In certain situations the delimiting pair of brackets, (...) is redundant, and may be omitted. However, in doubt you should enclose an *equation* within round brackets - they cannot harm.

expectation store Every *element* has at least one expectation associated with it. There are a number (usually at least two) of alternative storage areas for these expectations, corresponding to alternative specifications. Each such area is called an expectation store. Expectation store number one corresponds to the first expectation store, and so forth. The number of expectation stores is the same as the number of belief stores, and may be found by issuing the LOOK: command using the program argument. See also *b*elief store. The main commands which require expectations take them from the current default expectation store. This is initially store number one, but may be changed using the e argument to the CONTROL: command.

function A function is a scalar function defined by the F: command. Its value is not determined at the point where the F: command is used. Instead, the functional form is stored and evaluated therafter when called. Depending upon the arguments to the function at at a given time, the evaluation may change.

history file A file to which the command history is being saved. This file is only generated if the history argument to the KEEP: command has been used.

identifier Identifiers are the names used for the various quantities in [B/D]. Each identifier is a sequence of alphanumeric characters, beginning with an alphabetic character. The maximum length depends upon the machine implementation. Some identifiers may have as a suffix an index part, either known or unknown.

index Each index is either a single alphabetic character or an integer, each preceded by the period symbol ".". Integer indices are known and fixed. Character indices are allowed to vary (to take several different possible integer values), and are intended to become known at some future time.

index list The names of elements, functions, and so forth, may include one or more indices. The index parts of an element, function, etc., name always come after the main name. When defining quantities which include indices, you may not define a name which contains both known and unknown indices. However, an accompanying equation may.

label A *label* is an alphanumeric string starting with an alphabetic character, and prefixed with the @ symbol. [B/D] command lines can consist of such labels, possibly accompanied by commentative text, separated by one or more spaces from the label. You may define duplicate labels provided that they are on different input channels. If a duplicate label is found on the same file, the former is overwritten by the latter. The labels on a file may be viewed, with or without commentary, by using the LOOK: command with l and lall arguments.

log file A log of the outputs produced during a session may be retained in a log file.

multiple lines These consist of distinct command lines separated by the vertical line symbol |.

naming part Given a particular command, we call the part to the left hand side of the equals symbol the *naming* part, and the part to the right hand side the *definition* part of the construction. Thus here $Y.i.j$ is the naming part, and $m + a.i + b.j + e.i.j$ is the definition part of the construction.

ordering convention The following convention is adopted for the ordering of names. The initial part of the name, up to and excluding any digit, is considered first, and the basic ordering of the two names depends upon these parts. Thus 'x10' is considered to be 'left' of 'xa2' as 'x' is 'left' of 'xa'. Wherever the alphabetic parts are identical, the overall length is next taken into consideration, with shorter overall names to the left of longer names. Thus, 'x2' is to the left of 'x10'. Finally, when both the lengths and the alphabetic parts are the same, a straight alphabetical comparison is made between the two. Thus 'x11' is to the left of 'x12'.

previous adjustment The previous adjustment is only defined when two successive related adjustments (typically the first a simple adjustment and the second a partial adjustment) have taken place. In this case there will be available results concerning the difference between the two, representing the partial adjustment.

rounding For the purpose of conditional action, for example during use of the IF: command, we represent "true" by any real number whose value rounded to the nearest integer is unity, and "false" otherwise. A number which is exactly halfway between two whole numbers is rounded to the whole number with the greatest magnitude.

search path A search path tells [B/D] where it can find particular external files. Every search path typically includes the current directory as the first directory searched for a file. For macro input files, the default search path is set by the environment variable BDMACROS.

Bibliography

[1] P. S. Craig, M. Goldstein, A. H. Seheult, and J. A. Smith. Bayes linear strategies for history matching of hydrocarbon reservoirs. In J.-M. Bernardo et al., editors, *Bayesian Statistics 5*, pages 69–98, Oxford, 1996. University Press.

[2] B. de Finetti. Foresight:its logical laws, its subjective sources. *Annales de l'Institut Henri Poincaré*, 7, 1937.

[3] B. de Finetti. *Theory of probability, vol. 1*. Wiley, New York, 1974.

[4] B. de Finetti. *Theory of probability, vol. 2*. Wiley, New York, 1975.

[5] M. Farrow and M. Goldstein. Reconciling costs and benefits in experimental design. In J.-M. Bernardo et al., editors, *Bayesian Statistics 4*. Oxford University Press, 1992.

[6] M. Farrow and M. Goldstein. Bayes linear methods for grouped multivariate repeated measurement studies with application to crossover trials. *Biometrika*, 80(1):39–59, 1993.

[7] M. Farrow and M. Goldstein. Diagnostic geometry for Bayes linear prediction systems. In J.-M. Bernardo et al., editors, *Bayesian Statistics 5*, pages 561–568, Oxford, 1996. University Press.

[8] M. Goldstein. Approximate Bayesian inference with incompletely specified prior distributions. *Biometrika*, 61:629–631, 1974.

[9] M. Goldstein. Approximate Bayesian solutions to some nonparametric problems. *The Annals of Statistics*, 3:512–517, 1975.

[10] M. Goldstein. A note on some Bayesian nonparametric estimates. *The Annals of Statistics*, 3:736–740, 1975.

[11] M. Goldstein. Bayesian analysis of regression problems. *Biometrika*, 63:51–58, 1976.

[12] M. Goldstein. The variance modified linear Bayes estimator. *J. R. Statist. Soc.*, B:41:96–100, 1979.

[13] M. Goldstein. The linear Bayes regression estimator under weak prior assumptions. *Biometrika*, 67:621–628, 1980.

[14] M. Goldstein. Revising previsions: a geometric interpretation. *J. R. Statist. Soc.*, B:43:105–130, 1981.

[15] M. Goldstein. General variance modifications for linear Bayes estimators. *J. Amer. Statist. Ass.*, 78:616–618, 1983.

[16] M. Goldstein. The prevision of a prevision. *J. Amer. Statist. Ass.*, 78:817–819, 1983.

[17] M. Goldstein. Turning probabilities into expectations. *The Annals of Statistics*, 12:1551–1557, 1984.

[18] M. Goldstein. Temporal coherence. In J.-M. Bernardo et al., editors, *Bayesian Statistics 2*, pages 231–248. North Holland, 1985.

[19] M. Goldstein. Exchangeable belief structures. *J. Amer. Statist. Ass.*, 81:971–976, 1986.

[20] M. Goldstein. Separating beliefs. In P.K. Goel and A. Zellner, editors, *Bayesian Inference and Decision Techniques*. North Holland, Amsterdam, 1986.

[21] M. Goldstein. [B/D] introduction and overview. Technical Report 1:5, University of Hull Mathematics Research Reports, 1987.

[22] M. Goldstein. Can we build a subjectivist statistical package? In R. Viertl, editor, *Probability and Bayesian Statistics*, pages 203–217. Plenum, New York, 1987.

[23] M. Goldstein. Systematic analysis of limited belief specifications. *The Statistician*, 36:191–199, 1987.

[24] M. Goldstein. Adjusting belief structures. *J. R. Statist. Soc.*, B:50:133–154, 1988.

[25] M. Goldstein. The data trajectory. In J.-M. Bernardo et al., editors, *Bayesian Statistics 3*, pages 189–209. Oxford University Press, 1988.

[26] M. Goldstein. Influence and belief adjustment. In J.Q. Smith and R.M. Oliver, editors, *Influence Diagrams, Belief Nets and Decision Analysis.* Wiley, Chichester, 1990.

[27] M. Goldstein. Belief transforms and the comparison of hypotheses. *The Annals of Statistics*, 19:2067–2089, 1991.

[28] M. Goldstein. Technical specifications for the North West Water sewerage model. Technical report, Department of Mathematical Sciences, University of Durham, 1993.

[29] M. Goldstein. Belief revision: subjectivist principles and practice. In D. Prawitz and D Westerstahl, editors, *Logic and Philosophy of Science in Uppsala*, pages 117–130, 1994.

[30] M. Goldstein. Revising exchangeable beliefs: subjectivist foundations for the inductive argument. In P. Freeman and A. F. M. Smith, editors, *Aspects of Uncertainty: A Tribute to D. V. Lindley.* Wiley, 1994.

[31] M. Goldstein. Bayes linear methods I - Adjusting beliefs: concepts and properties. Technical Report 1995/1, Department of Mathematical Sciences, University of Durham, 1995.

[32] M. Goldstein, M. Farrow, and T. Spiropoulos. Prediction under the influence: Bayes linear influence diagrams for prediction in a large brewery. *The Statistician*, 42(2):445–459, 1993.

[33] M. Goldstein and A. O'Hagan. Bayes linear sufficiency and systems of expert posterior assessments. *J. R. Statist. Soc. B*, 58(2):301–316, 1996.

[34] M. Goldstein and D. A. Wooff. Robustness measures for Bayes linear analyses. *Journal of Statistical Planning and Inference*, 40(2-3):261–277, 1994.

[35] M. Goldstein and D. A. Wooff. Bayes linear computation: concepts, implementation and programming environment. *Statistics and Computing*, 5:327–341, 1995.

[36] M. Goldstein and D. A. Wooff. Choosing sample sizes in balanced experimental designs: a Bayes linear approach. *The Statistician*, 46(2):167–183, 1997.

[37] M. Goldstein and D. A. Wooff. Adjusting exchangeable beliefs. *Biometrika*, 85(1):39–54, 1998.

[38] G. H. Gonnet and R. Baeza-Yates. *Handbook of algorithms and data structures in Pascal and C.* Addison-Wesley, 1991.

[39] D. E. Knuth. *The art of computer programming*, volume 1: Fundamental algorithms. Addison-Wesley, Reading, Massachusetts, 2 edition, 1973.

[40] D. E. Knuth. *The art of computer programming*, volume 2: Seminumerical algorithms. Addison-Wesley, Reading, Massachusetts, 2 edition, 1981.

[41] A. O'Hagan. Robust modeling for asset management. *Journal of Statistical Planning and Inference*, 40(2-3):245–255, 1994.

[42] J. Q. Smith and R. M. Oliver, editors. *Influence Diagrams, Belief Nets and Decision Analysis.* Wiley, Chichester, 1990.

[43] D. J. Wilkinson and M. Goldstein. Bayes linear adjustment for variance matrices. In J.-M. Bernardo et al., editors, *Bayesian Statistics 5*, pages 791–800, Oxford, 1996. University Press.

[44] D. J. Wilkinson and M. Goldstein. Bayes linear covariance matrix adjustment for multivariate dynamic linear models. *In submission*, 1996.

[45] Niklaus Wirth. *Algorithms and data structures.* Prentice-Hall, London, 1986.

[46] D. A. Wooff. [B/D] reference manual. Technical Report 1:6, University of Hull Mathematics Research Reports, 1987.

[47] D. A. Wooff. [B/D] works. In J.-M. Bernardo et al., editors, *Bayesian Statistics 4*, pages 851–859. Oxford University Press, 1992.

[48] D. A. Wooff. Bayes linear methods II - An example with an introduction to [B/D]. Technical Report 1995/2, Department of Mathematical Sciences, University of Durham, 1995.

[49] D. A. Wooff and M. Goldstein. Bayes linear methods III - analysing Bayes linear influence diagrams and Exchangeability in [B/D]. Technical Report 1995/3, Department of Mathematical Sciences, University of Durham, 1995.

Index