ConQuest Command Reference

This reference document supplements and updates Section 4 of the ConQuest manual[1]. This command reference is regularly updated and contains the latest general information about the syntax of ConQuest command statements followed by an alphabetical reference of ConQuest commands.

[1] ACER ConQuest version 2.0: generalised item response modelling software (ISBN 9780864314543).

Contents

This document contains general information about the syntax of ConQuest command statements, followed by an alphabetical reference of ConQuest commands.

All ConQuest commands can be accessed through a command line interface. In addition, the majority of commands with their options can be accessed through the graphical user interface. This document describes the syntax for the command line interface and the graphical user interface accessibility of each of these commands.

COMMAND STATEMENT SYNTAX

A ConQuest statement can consist of between one and five components: a command, arguments, options, an Outdirection and an Indirection. The general syntax of a ConQuest statement is as follows:

```
Command Arguments ! Options >> Outdirection << Indirection;
```

The first text in a statement must be a command. The command is followed by an argument with a space used as a separator. Some commands have optional arguments; others require an argument. An exclamation mark (`!`) separates arguments from options; if there is no argument, the exclamation mark can separate a command from an option. The characters << or >> separate a file redirection (either an indirection or an outdirection) from the preceding elements of the statement.

ConQuest syntax has the following additional features:

(1) A statement must be terminated with a semi-colon (`;`). The semi-colon, not the `return` or `new line` character, is the separator between statements.

(2) You can type more than one statement on a line. However, pressing the Enter key after each statement will make the statements easier to read.

(3) A statement can be 3072 characters in length and can cover any number of lines on the screen or in a command file. No continuation character is required.

(4) Comments are placed between /* and */. They can appear anywhere in a command file, and their length is unlimited. Comments cannot be nested inside another comment.

(5) The command language is not case sensitive.

(6) The order in which command statements can be entered into ConQuest is not fixed. There are, however, logical constraints on the ordering. For example, `show` statements cannot precede the `estimate` statement, which in turn cannot precede the `model`, `format` or `datafile` statements, all three of which must be provided before ConQuest can analyse a data set.

(7) Any command file that you want ConQuest to read must be an ASCII text file. If you create a command file, a data file or a design matrix file with a word processor, remember to save the file as text only. Do not use 'typesetter', or 'curly', double quotation marks (“ ”), as these are not the same ASCII characters as 'straight' quotation marks (" ").

(8) User-provided variable names must begin with an alphabetic character and must be made up of alphabetic characters or digits. Spaces are not allowed in variable names and some characters and names are reserved for ConQuest use (see ***List of illegal characters and words for variable names*** at the end of this document).

(9) All commands, as well as arguments and options that consist of ConQuest reserved words, can be abbreviated to their shortest unambiguous root. For example, the following are all valid:

```
caseweight, caseweigh, caseweig, casewei, casewe, casew,
case, cas, ca
codes, code, cod, co
converge=, converg=, conver=, conve=, conv=, con=, co=
datafile, datafil, datafi, dataf, data, dat, da, d
estimate, estimat, estima, estim, esti, est, es
export, expor, expo, exp, ex
```

Example Statements

```
codes 0,1,2;
```

`codes` is the command, and the argument is `0,1,2`.

```
format responses 11-20 ! rater(2),essay(5);
```

`format` is the command, `responses 11-20` is the argument, and `rater(2)` and `essay(5)` are the options.

```
show ! cases=eap >> file.out;
```

`show` is the command, there is no argument, `cases=eap` is the option, and `>> file.out` is the redirection.

TOKENS AND MATRIX VARIABLES

A `token` is an alphanumeric string that is contained between % characters (for example, %xyz%). A value can be set to a token through the use of a ConQuest `let` command. When the token is detected in subsequent syntax it is replaced by the value it represents. Tokens can be used in any context.

The `token` date is created automatically, and it is available as (%date%) at any time.

A `variable` is a matrix value that is set through a compute command or created, when requested, by a ConQuest procedure. A variable can be used in a compute command, or produced by a compute command. A variable can also be used as input in a number of procedures. A variable cannot be used as a component of the command language.

A number of analysis routines can be directed to save their results as variables – typically sets of matrices. These variables can be subsequently manipulated, saved or plotted.

Example Statements

```
let n=10;
generate ! nitems=%n%;
```

Assigns the string '10' to the token 'n' so that when the subsequent `generate` command is executed the string %n% is replaced by the string '10'.

```
compute n=10;
compute m=n+2;
print m;
```

Assigns the value '10' to the variable 'n', adds two to n and produces 'm' and then prints the value of m (ie 12).

```
let n=matrix(2:2);
compute n={10,-23,0.4,1};
compute m=inv(n);
print n,m;
```

'n' is created as a 2 by 2 matrix which is populated with the four values between the braces, the inverse of 'n' is then calculated and saved as 'm', finally the values of 'n' and 'm' are printed.

```
estimate ! matrixout=r;
compute fit=r_itemfit[,3];
plot r_itemparams fit;
print r_estimatecovariances ! filetype=xlsx >> covariances.xlsx;
```

Estimation is undertaken and a set of matrices containing results is created (see estimate command). The item parameter estimates are plotted against the unweighted mean square fit statistics and then the parameter estimate covariance matrix is saved as an excel file.

Loops and Conditional Execution

Loops and conditional execution of control code can be implemented through the use of the `for`, `while`, `if`, `dofor` and `doif` commands. The `for`, `while`, and `if` are commands are typically used to manipulate matrix variables and their contents. `dofor` and `doif` are typically used to loop over token values or conditionally execute code based upon tokens.

Explicit and Implicit Variables

When ConQuest reads data from a file identified with the `datafile` command with a structure as described by the `format` command variables of two different types can be generated. Explicit variables are variables that are listed in a `format` statement. Implicit variables are variables that are associated with specific columns in the data file referred to in the format statements as responses. For a full illustration of these two classes of variables see the `format` command.

Using ConQuest Commands

ConQuest is available with both a graphical user interface (GUI) and a command line, or console, interface. The ConQuest command statement syntax used by the GUI and the console versions is identical. In general, the console version runs faster than the GUI version, but the GUI version is more user friendly. GUI version and console version system files are fully compatible.

Entering Statements via the Console Interface

When the console version of ConQuest is started, the less than character (<) is displayed. This is the ConQuest prompt. When the ConQuest prompt is displayed, any appropriate ConQuest statement can be entered.

As with any command line interface, ConQuest attempts to execute the statement when you press the Enter key. If you have not yet entered a semi-colon (`;`) to indicate the end of the statement, the ConQuest prompt changes to a plus sign (`+`) to indicate that the statement is continuing on a new line.

On many occasions, a file containing a set of ConQuest statements (ie a ConQuest command file) will be prepared with a text editor, and you will want ConQuest to run the set of statements that are in the file. If we suppose the ConQuest command file is called `myfile.cqc`, then the statements in the file can be executed in two ways.

In the first method, start ConQuest and then type the statement

```
submit myfile.cqc;
```

A second method, which will work when running from a command-line interpreter (cmd), is to provide the command file as a command line argument. You launch ConQuest and provide the command file in one step using

```
ConQuest4CMD myfile.cqc;
```

With either method, after you press the Enter key, ConQuest will proceed to execute each statement in the file. As statements are executed, they will be echoed on the screen. If you have requested displays of the analysis results and have not redirected them to a file, they will be displayed on the screen.

Entering Commands via the GUI Interface

Once you have launched the GUI interface (double-click on ConQuest4GUI.exe), you can type command statements or open a command file in the GUI input window and then select Run➔Run All. In addition, the GUI interface has menu selections that will build and execute ConQuest command statements. Access to the commands with the GUI is described in the following for each command.

COMMANDS

The remainder of this document describes the ConQuest commands. The arguments or options that are listed below the commands are reserved words when used with that command.

caseweight

Specifies an explicit variable that is to be used as a case weight.

Argument

An explicit variable that is used as a case weight.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
caseweight pweight;
```

The explicit variable pweight contains the weight for each case.

```
caseweight;
```

No case weights are used.

GUI Access

Command ➔ Case Weight.

Select the Case Weight menu item. The radio button allows case weighting to be toggled. If cases are to be weighted then a variable must be selected from the candidate list of explicit variables.

Notes

(1) The caseweight statement stays in effect until it is replaced with another caseweight statement or until a reset statement is issued. If you have run a model with case weights and then want to remove the case weights from the model, the simplest approach is to issue a caseweight statement with no arguments.

(2) A variable that will be a case weight must be listed in the format as an explicit variable.

(3) Case weighting is applied to item response model estimation, but not to traditional or descriptive statistics.

categorise

Sets up a dummy codes for a categorical regression variable.

Argument

The argument takes the form *var(v1:v2:…:vN)* Where *var* is a categorical variable and *v1:v2:…:vN* is a list of values that give levels of the categorical variable. A set of N-1 new dichotomously coded variables are created to represent the N categories of the original variable.

If the values that represent levels of the categorical variables contain leading or trailing spaces then the values will need to be enclosed in quotes. If observed levels are omitted from the list they are treated as missing data.

When *var is specified* as a regression variable it will be replaced by the *N*-1 variables *var_1, var_2, var_(N-1).*

The variables *var_1, var_2, var_(N-1)* cannot be accessed directly by any command.

Options

`Coding method`

Specifies the type of dummy coding. It can be one off `dummy`, or `effect`. The default is *`dummy`*. The first category is used as reference category.

Redirection

Redirection is not applicable to this command.

Examples

```
categorise gender(M:F);
```

Establishes "M" as a reference category so M will be coded "0" and F will be coded "1".

```
categorise grade(3:4:5:6:7) ! effect;
```

Establishes four variables to represent the five response categories for `grade`. Effect coding is used and the reference category is "3".

`grade=3` corresponds to `grade_1=-1`, `grade_2=-1`, `grade_3=-1`, and `grade_4=-1`.

`grade=4` corresponds to `grade_1=1`, `grade_2=0`, `grade_3=0`, and `grade_4=0`.

`grade=5` corresponds to `grade_1=0`, `grade_2=1`, `grade__3=0`, and `grade_4=0`.

`grade=6` corresponds to `grade_1=0`, `grade_2=0`, `grade_3=1`, and `grade_4=0`.

`grade=7` corresponds to `grade_1=0`, `grade_2=0`, `grade_3=0`, and `grade_4=1`.

```
Categorise size (S:M:L);
```

Establishes two variables to represent the three response categories for `size` (Small, Medium and Large). Dummy coding is used and the reference category is “S”.

`size=S` corresponds to `size_1=0`; and `size_2=0`;

`size=M` corresponds to `size_1=1`; and `size_2=0`;

`size=L` corresponds to `size_1=0`; and `size_2=1`;

GUI Access

Access to this command through the GUI is not available.

Notes

(1) Any levels of the variable that are omitted from the code list are treated as missing data.

(2) To alter the reference level change the order in which the levels are listed.

(3) Only one variable can be processed with a categorise command. Use multiple commands to categorise multiple variables.

clear

Removes variables or tokens from your workspace.

Argument

A comma separated list of variables and/or tokens or one of `all`, `tokens` or `variables`. The default is *`all`*.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
clear all;
```

Clears all variables and tokens from your workspace.

```
clear x, date;
```

Deletes the variable (or tokens) x and date from your workspace.

GUI Access

Workspace ➔ Tokens and Variables.

Results in a dialog box. The box displays the list of available tokens and variables. The Clear All or Clear Selected buttons can be used either to clear all objects listed in the box or clear the selected objects, respectively. The action takes immediate place once the button is clicked.

Notes

(1) The work space is your temporary working environment and includes any user-defined elements (tokens and matrices). Elements can be saved using the print command, otherwise they will be deleted when closing ConQuest.

codes

Lists the characters that are to be regarded as valid data for the responses.

Argument

A comma-delimited or space-delimited list of response codes.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
codes 0,1,2,3;
```

The valid response codes are 0, 1, 2 and 3.

```
codes a b c d;
```

The valid response codes are a, b, c and d.

```
codes 1, 2, 3, 4, 5, " ";
```

The valid response codes are 1, 2, 3, 4, 5, and a blank.

```
codes " 1", " 2", " 3", "10";
```

Each response code takes two columns. The first three that are listed have leading spaces, which must be included.

GUI Access

Command ➔ Codes.

The list of codes must be entered using the same syntax guidelines as described above for the `codelist`.

Notes

(1) If a blank is to be used as a valid response code or if a blank is part of a valid response code, double quotation marks (`" "`) must surround the response code that includes the blank.

(2) *`Codelist`* specifies the response codes that will be valid after any recoding has been performed by the `recode` statement.

(3) If a *`codes`* statement is provided, then any character found in the response block of the data file (as defined by the format statement) and not found in *`codelist`* will be treated as missing-response data.

(4) Any missing-response codes (as defined by the *`set`* command argument *`missing`*) in *`codelist`* will be ignored. In other words, *`missing`* overrides the `codes` statement.

(5) If a `codes` statement is not provided, then all characters found in the response block of the data file, other than those specified as missing-response codes by the *set* command argument *`missing`*, will be considered valid.

(6) The number of response categories modelled by ConQuest is equal to the number of unique response codes (after recoding).

(7) Response categories and item scores are *not* the same thing.

compute

Undertakes mathematical computations and creates a ConQuest data object to store the result. The data object can be a real number or a matrix. The command word `compute` is optional and is assumed if a command word is omitted.

Argument

```
t=mathematical expression
or
t={list of values}
```

A comma separated list of real numbers that are used to populate an existing matrix. Columns cycle fastest.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
compute x=10;
x=10;
```

Alternatives for creating a 1-by-1 matrix with x(0,0)=10.

```
compute x={1,2,3,4};
x={1,2,3,4};
```

Either form populates a pre-existing matrix with these values. Columns cycle fastest, so the result is x(1,1)=1, x(1,2)=2, x(2,1)=3, and x(2,2)=4. A matrix with as many elements as given numbers must have been pre-defined via a `let` command.

```
compute x=a+b;
x=a+b;
```

Alternatives for creating the matrix x as matrix sum of matrices a and b.

```
compute m[10,3]=5;
m[10,3]=5;
```

Either form sets the row=10, column=3 element of the matrix m to 5.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) The available functions and operators are listed and described in the section headed ***compute command operators and functions***.

(2) Parentheses can be nested to 10 deep.

(3) To populate a matrix with a set of values that matrix must be previously defined using the `let` command. If the right hand side of the assignment ('=') is a matrix or mathematical expression, then the output matrix need not be defined in advance.

(4) Sub matrices can be extracted from matrices by appending [*rowstart:rowend,colstart:colend*] to the name of a matrix variable. If all rows are required *rowstart* and *rowend* can be omitted. If all columns are required *colstart* and *colend* can be omitted. If a single row is required *rowend* and the colon ":" can be omitted. If a single column is required *colend* and the colon ":" can be omitted.

(5) Single elements of a matrix can be specified to the left of the equal operator '=' by appending [*row,col*] to the name of a matrix variable. Sub matrices cannot be specified to the left of the equal operator '='.

(6) Tokens can be used in any context. Variables however can only be used in a `compute`, `print` or `scatter` command and as matrix input or matrix output for commands that accept such input and output.

datafile

Specifies the name, location and type of file containing the data that will be analysed.

Argument

`*filename*`

The name or pathname (in the format used by the host operating system) that contains the data to be analysed. The file type can be ASCII text file or SPSS system file.

Options

A list of comma-separated options. Each option is discussed below.

`filetype=type`

Indicates the format of the datafile. Available options are spss and text. The default is *text*. If an input file has SPSS format then a `format` command is automatically generated by ConQuest.

`responses=`*`varlist`*

A space delimited list of variables from the SPSS file that are to be used as the (generalised) item responses in the model. The keyword ‘`to`’ can be used to specify a list of sequentially placed variables in the SPSS file. This option is only applicable for SPSS input files.

`facets=`*`string`*

Describes the implicit variables that underlie the responses (see `format` command).

`keeps=`*`varlist`*

A space delimited list of additional variables read from the SPSS file and retained as explicit variables. The keyword ‘to’ can be used to specify a list of sequentially placed variables in the SPSS file. This option is only applicable for SPSS input files.

`weight=`*`var`*

A variable from the SPSS file to be used as a caseweight variable. The default is no caseweight variable.

`pid=`*`var`*

A variable from the SPSS file to be used as a case identifier. The default is no `pid`. See `format` for a description of the `pid` variable.

`width=`*`n`*

A value to use as the width of the response variables. The *n* left most characters of the SPSS response variables are retained and used as the (generalised) item responses. The default width is *1*.

Redirection

`<< filename`

The name or pathname (in the format used by the host operating system) of the ASCII text file or SPSS system file that contains the data to be analysed.

The specification of the filename as an argument or as a redirection are alternatives.

`>> outfilename`

An optional file name for a text file version of the data. The outfile is used in conjunction with the `filetype=spss` option and results in a text copy of the analysed data being retained.

Examples

```
datafile mydata.txt;
```

The data file to be analysed is called `mydata.txt`, and it is in the same directory as the ConQuest application.

```
datafile \math\test1.dat;
```

The data file to be analysed is called `test1.dat`, and it is located in the directory `math`.

```
datafile << c:\math\test1.dat;
```

The data file to be analysed is called `test1.dat`, and it is located in the directory `math` on the C: drive.

```
datafile test2.sav ! filetype=spss, responses=item1 to
   item16, keeps=country, weight= pwgt, facets=tasks(16),
   pid=id >> test.dat;
```

The data file to be analysed is called `test2.sav`, and it is an SPSS file. The set of SPSS variables beginning with `item1` and concluding with `item16` are retained as responses, country is retained as an explicit variable, `pwgt` will be used as a `caseweight` and `id` as a `pid`. The responses will be referred to as `tasks`. The requested data will be written to the file `test.dat` and it will be retained after the analysis.

Use of this datafile command is equivalent to the following three commands:

```
datafile << test2.dat;
format pid 1-15 responses 16-31(a1)
  pwgt32-42 country 42-51 ! tasks(16);
caseweight pwgt;
```

```
datafile test2.sav ! filetype=spss, responses=item1 to
   item16, keeps=country, weight=pwgt, facets=tasks(16),
   pid=id;
```

This example is equivalent to the previous example except that the requested data will be written to a scratch file that will not be retained after the analysis.

GUI Access

Command ➔Data File.

Note that GUI access does not yet support SPSS file imports.

Notes

(1) The actual format of `file name` will depend upon the host operating system. For example, under the VMS operating system, the second example may have to be written as

```
datafile [.math]test1.dat;
```

(2) When inputting the response data in a data file, remember that ConQuest treats blanks and periods found in the responses as missing-response data unless you either use a `codes` statement to specify that one or both are to be treated as valid response codes, or use the `set` command argument missing to change the missing-response code.

(3) The layout of your data file lines and records must conform to the rules of the `format` command.

(4) A file of simulated data can be created with the `generate` command.

(5) When using SPSS files, both character and numeric variables can be used. The conversion for use by ConQuest of numeric variables is governed by the width and decimals properties of the variables in the SPSS file. However, if ConQuest uses the converted variables as strings, a leading blank will be added. This needs to be accounted for when specifying particular values for example in the `keep` and `drop` options of various command statements.

(6) The maximum width of a variable read from an SPSS files is 256 characters

(7) System missing numeric values in SPSS are converted to a period character (.) in a field of width set by the width property in the SPSS file.

(8) If using variables that are treated as string, for example in `group` statement, is recommended to convert the type to String within SPSS before running in ConQuest.

delete

Omit data for selected implicit variables from analyses.

Argument

This command has no argument.

Options

A list of implicit variables and the levels that are to be omitted from the analysis for each variable.

Redirection

Redirection is not applicable to this command.

Examples

```
delete ! item (1-10);
```

Omits items 1 through 10 from the analysis.

```
delete ! rater (2, 3, 5-8);
```

The above example omits data from raters 2, 3, 5, 6, 7, and 8 from the analysis.

GUI Access

Command ➔ Delete.

The list of candidate implicit variables is listed in the list box. Multiple selections can be made by shift-clicking.

Notes

(1) `delete` statement definitions stay in effect until a `reset` statement is issued.

(2) `delete` preserves the original numbering of items (as determined by the `format` and the `model` statements) for the purposes of data display and for labels. Note however that it does change parameter numbering. This means that anchor and initial values files may need to be modified to reflect the parameter numbering that is altered with the any deletions.

(3) To omit data for specified values of explicit variables the missing data command can be used.

(4) See the `dropcases` and `keepcases` commands which are used to limit analysis to a subset of the data based on explicit variables.

descriptives

Calculates a range of descriptive statistics for the estimated latent variables.

Argument

This command does not have an argument.

Options

`estimates=`*`type`*

`type` can be `eap`, `pv`, `mle` or `wle`. If `estimates=`*`eap`*, the descriptive statistics will be constructed from expected a-posteriori values for each case; if `estimates=`*`pv`*, the descriptive will be constructed from plausible values for each case; if `estimates=`*`mle`*, the descriptive statistics will be constructed from maximum likelihood cases estimates and if `estimates=`*`wle`*, the descriptive statistics will be constructed from weighted likelihood cases estimates.

`group=`*`variable`*

An explicit variable to be used as grouping variable. Results will be reported for each value of the group variable. The variable must have been listed in a previous `group` command.

`percentiles=`*`n1:n2:…:ni`*

`ni` is a requested percentile to be computed.

`cuts=`*`n1:n2:…:ni`*

Requests calculation of the proportion of students that lie within a set of intervals on the latent scale. *`ni`* is a requested cut point. The specification of *`i`* cut points results in *`i+1`* intervals.

`bench=`*`n1:n2:n3`*

Requests calculation of the proportion of students that lie either side of a benchmark location on the latent scale. *`n1`* is the benchmark location, *`n2`* is the uncertainty in the location, expressed as standard deviation and *`n3`* is the number of replications to use to estimate the standard error of the proportion of students above and below the benchmark location.

`filetype=`*`type`*

`type` can take the value *`xls`*, *`xlsx`*, *`excel`* or *`text`*. It sets the format of the results file. Both *`xls`* and *`excel`* create files readable by all versions of Excel. The *`xlsx`* format is for Excel 2007 and higher. The default is *`text`*. If no redirection file is provided this option is ignored.

`matrixout=`*`name`*

`name` is a matrix (or set of matrices) that will be created and will hold the results in your workspace. Any existing matrices with matching names will be overwritten without warning. The content of the matrices

is described in the section ***Matrix Objects Created by Analysis Commands***.

`display=`*`reply`*

If *`reply`* is *`short`*, results will not be displayed for individual plausible values .The default is *`long`*.

Redirection

```
>> filename
```

`filename` is the name of a file to which results can be written.

Examples

```
descriptives ! estimates=pv;
```

Using plausible values produces the mean, standard deviation and variance (and the associated error variance) for each of the latent dimensions.

```
descriptives ! estimates=pv, group=gender;
```

Using plausible values produces the mean, standard deviation and variance (and the associated error variance) for each of the latent dimensions for each value of gender.

```
descriptives ! estimates=mle, percentiles=10:50:90;
```

Using maximum likelihood estimates produces the mean, standard deviation and variance (and the associated error variance) for each of the latent dimensions. The 10th, 50th and 90th percentiles are also estimated for each dimension.

```
descriptives ! estimates=pv, cuts=-0.5:0.0:0.5;
```

Using plausible values estimates produces the mean, standard deviation and variance (and the associated error variance) for each of the latent dimensions. The proportion of students in the four intervals: less than –0.5; between –0.5 and 0.0; between 0.0 and 0.5; and greater than 0.5 are also estimated for each dimension.

```
descriptives ! estimates=pv, bench=-1.0:0.1:1000;
```

Using plausible values estimates produces the mean, standard deviation and variance (and the associated error variance) for each of the latent dimensions. The proportion of students above and below a benchmark of –1.0 is also estimated for each dimension. The error in these proportions is based upon an uncertainty of 0.1 in the benchmark location. The error was estimated using 1000 replications.

GUI Access

Analysis ➔ Descriptives ➔ Latent Variables.

Notes

(5) The ability estimates requested (`wle`, `mle`, `eap` and `latent`) must have been previously estimated (see `show` command).

directory

Displays the name of the current working directory. The working directory is where ConQuest looks for files and writes files when a full directory path is not provided as part of a file specification.

Argument

This command does not have an argument.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
directory;
```

GUI Access

Access to this command through the GUI is not available.

Notes

(1) For the GUI version of ConQuest the result of this command is shown in the status bar.

(2) To change the working directory see the `set` argument option `directory`.

dofor

Allows looping of syntax.

Argument

A list of comma-separated arguments. If an element in the list takes the form *i1-i2* where *i1* and *i2* are integers then it is expanded to be a list of all integers from *i1* to *i2* inclusive of *i1* and *i2.*

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
dofor x=M,F;
   Plot icc ! group=gender; keep=%x%;
enddo;
```

Produces plots for students with gender value M and then gender value F.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) `dofor` must be used in conjunction with `enddo`.

(2) `dofor` loops cannot be nested.

(3) See the command `for` for alternative looping options.

doif

Allows conditional execution of syntax.

Argument

`argument` *`logical condition`*

If logical condition evaluates to `true`, the set of ConQuest commands is executed. The commands are not executed if the logical condition does not evaluate to `true`.

The logical condition can be true, false or of the form *s*1 operator *s*2, where *s*1 and *s*2 are strings and operator is one of the following

```
==    equality
=>    greater than or equal to
>=    greater than or equal to
=<    less than or equal to
<=    less than or equal to
!=    not equal to
>     greater than
<     less than
```

For each of *s*1 and *s*2 ConQuest first attempts to convert it to a numeric value. If *s*1 is a numeric value the operator is applied numerically. If not, a string comparison occurs between *s*1 and *s*2.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
doif %x%==M;
   Plot icc ! group=gender; keep=%x%;
endif;
```

Produces plots for students with gender value M.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) `doif` must be used in conjunction with `endif`.

(2) `doif` conditions cannot be nested.

(3) See the command `if` for alternative conditional execution options.

dropcases

List of values for explicit variables that if matched will cause a record to be omitted from the analysis.

Argument

`list of drop codes`

The `list of drop codes` is a comma separated list of values that will be treated as drop values for the subsequently listed explicit variable(s).

When checking for drop codes two types of matches are possible. EXACT matches occur when a code in the data is compared to a drop code value using an exact string match. A code will be regarded as a drop value if the code string matches the drop string exactly, including leading and trailing blank characters. The alternative is a TRIM match that first trims leading and trailing spaces from both the drop string and the code string and then compares the results.

The key words `blank` and `dot`, can be used in the drop code list to ensure TRIM matching of a blank character and a period. Values in the list of codes that are placed in double quotes are matched with an EXACT match. Values not in quotes are matched with a TRIM match.

Options

A comma separated list of explicit variables.

Redirection

Redirection is not applicable to this command.

Examples

```
dropcases blank, dot, 99 ! age;
```

Sets blank, dot and 99 (all using a trim match) as drop codes for the explicit variable `age`.

```
dropcases blank, dot, " 99" ! age;
```

Sets blank, and dot (using a trim match) and 99 with leading spaces (using an exact match) as drop codes for the explicit variable `age`.

```
dropcases M ! gender;
```

Sets M as a drop code for the explicit variable `gender`.

GUI Access

Command ➔ Drop Cases.

Select explicit variables from the list (shift-click for multiple selections) and choose the matching drop value codes. The syntax of the drop code list must match that described above for `list of drop codes.`

Notes

(1) Drop values can only be specified for explicit variables.

(2) Complete data records that match drop values are excluded from all analyses.

(3) If multiple records per case are used in conjunction with a `pid`, then the `dropcases` applies at the record level not the case level.

(4) See the `missing` command which can be used to omit specified levels of explicit variables from an analysis and the `delete` command which can be used to omit specified levels of implicit variables from an analysis.

(5) See the `keepcases` command which can be used to keep specified levels of explicit variables in the analysis.

(6) When used in conjunction with SPSS input, note that character strings may include trailing or leading spaces and this may have implications for appropriate selection of a match method.

else

Used as part of a `doif` condition.

Argument

This command does not have an argument.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
doif %x%==M;
   print "Plot for Males";
   plot icc ! group=gender; keep=M;
else;
   print "Plot for Females";
   plot icc ! group=gender; keep=F;
endif;
```

Produces plots for students with gender value M or F depending upon the value of the token %x%.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) `else` must be used in conjunction with `doif` and `endif`.

enddo

Terminates a `dofor` loop.

Argument

This command does not have an argument.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
dofor x=M,F;
   plot icc ! group=gender; keep=%x%;
enddo;
```

Produces plots for students with gender value M and then gender value F.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) `enddo` must be used in conjunction with `dofor`.

endif

Terminates a `doif` condition.

Argument

This command does not have an argument.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
doif %x%=M;
   plot icc ! group=gender; keep=%x%;
endif;
```

Produces plots for students with gender value M.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) `endif` must be used in conjunction with `doif`.

equivalence

Instructs ConQuest to produce a raw score to ability estimate equivalence.

Argument

`estimate type`

`Estimate type` must be either `wle` or `mle`

Options

`option list` A list of comma-separated options. Each option is discussed below.

`matrixin=`*`name`*

`name` is an existing matrix than can be used as source for the item parameter values.

`matrixout=`*`name`*

`name` is a matrix that will be created and will hold the results. It will be matrix with three columns and as many rows as there are score point. Column 1, contains the score value, column 2 the matching maximum likelihood estimate and 3 contains the standard error. More detail on the content of the matrices is described in the section ***Matrix Objects Created by Analysis Commands***.

`display=` *`reply`*

If *`reply`* is `no`, results will not be displayed. The default is `yes`.

Redirection

`>>` *`filename`* A file name for output.

`<<` *`filename`* A file name of item parameters.

Examples

```
equivalence wle;
```

Produces a raw score to weighted likelihood estimate equivalence table.

```
equivalence mle  >> mle.txt;
```

Produces a raw score to maximum likelihood estimate equivalence table and save it in the file mle.txt.

GUI Access

Tables➔Raw Score <-> Logit Equivalence➔MLE

Tables➔Raw Score <-> Logit Equivalence➔WLE

Tables→Raw Score <-> Logit Equivalence File→MLE

Tables→Raw Score <-> Logit Equivalence File→WLE

Notes

(1) The equivalence table assumes a complete response vector and integer scoring.

(1) Maximum and minimum values for maximum likelihood values are set using the `perfect/zero=` option of the `set` command

(2) If an input file is not specified then a model must have been estimated and the table is provided for the current model.

(3) If an input file is specified then equivalence table can be requested at any time

(4) An input file must be an ASCII file containing a list of item parameter estimates. Each line of the file should consist of the information for a single parameter with the item parameters being supplied in the Andrich delta plus tau format. Each line of the file should contain three values: item number, category number and the parameter value. The item difficulty parameter is signified by a category number of zero.

For example to indicate 3 dichotomous items the file could look as follows:

```
1  0  0.6
2  0  -1.5
3  0  2.3
```

To indicate 3 items each with three response categories the file could look as follows:

```
1  0  0.6
1  1  -0.2
2  0  -1.5
2  1  -0.5
3  0  2.3
3  1  1.1
```

Note that the order of the parameters does not matter and there is one fewer category parameter than there is categories. The last category parameter is assumed to be the negative sum of those provided.

(5) An input matrix must contain three columns. Each row of the matrix should consist of the information for a single parameter with the item parameters being supplied in the Andrich delta plus tau format. Column 1 is the item number, column 2, the category number and column 3 the parameter value. The item difficulty parameter is signified by a category number of zero.

(6) An input matrix cannot be used at the same time as an input file.

estimate

Begin estimation.

Argument

This command does not have an argument.

Options

A list of comma-separated options. Each option is discussed below.

`method=`*`type`*

Indicates the type of numerical integration that is to be used. *`type`* can take the value *`gauss`*, *`montecarlo`*, *`quadrature`* or *`JML`*. The default is *`gauss`*.

`nodes=`*`n`*

Specifies the number of nodes that will be used in the numerical integration. If the *`quadrature`* or *`gauss`* method has been requested, this value is the number of nodes to be used for each dimension. If the *`montecarlo`* method has been selected, it is the total number of nodes. The default value is *`15`* per dimension if the method is *`gauss`* or *`quadrature,`* and 1000 nodes in total if the method is *`montecarlo`*. The `nodes` option is ignored if `method=JML`.

`convergence=`*`f`*

Instructs estimation to terminate when the largest change in any parameter estimate between successive iterations of the EM algorithm is less than *`f`*. The default value is *`0.0001`*.

`iterations=`*`n`*

Specifies the maximum number of iterations for the EM algorithm. Estimation will terminate when either the iteration criterion or the convergence criterion is met. The default iterations value is *`200`*.

`storage=`*`type`*

Indicates whether the estimation temporary file will be created on disk or stored in random access memory. *`type`* can be the value *`ram`* or *`disk`*. The default is *`disk`* and is generally recommended. Estimation may be faster if *`ram`* is chosen, but if the data file is large and your computer's random access memory is limited, it is best to use *`disk`*.

`minnode=`*`f`*

Sets the minimum node value when using the *`quadrature`* method. The default is *`-6.0`*. All other methods ignore this option.

`maxnode=`*`f`*

Sets the maximum node value when using the *`quadrature`* method. The default is *`6.0`*. All other methods ignore this option.

`plausible=`*`filename`*

When the `plausible` option is specified, a file containing plausible values and EAP estimates for all cases will be created and named *`filename`*. Creating a plausible values file causes any distributions requested by a `show` statement to be produced more quickly. The default is to create no plausible value file.

The default number of plausible values generated is five. To alter this default, use the `set` command argument `n_plausible`.

If we use *np* to indicate the number of plausible values that are drawn for each case and let *nd* be the number of dimensions in the model, then the format of the plausible value file will be as follows.

It will contain *np*+3 lines per case.

Line 1 will contain the case number in columns 1 through 5.

Line 2 to line *np*+1 will each contain *nd* plausible values in the format *nd*(t13, *nd*(f6.2)).

Line *np*+2 will contain *nd* EAP estimates in the format *nd*(f10.5, 1x).

Line *np*+3 will contain *nd* posterior variance estimates in the format *nd*(f10.5, 1x).

`stderr=`*`type`*

Specifies how or whether standard errors are to be calculated. *`type`* can take the value *`quick`*, *`empirical`* or *`none`*. *`empirical`* is the default and uses empirical differentiation of the likelihood. While this method provides the most accurate estimates of the asymptotic error variances that ConQuest can compute, it may take a considerable amount of computing time, even on very fast machines. *`quick`* standard errors are suitable when dichotomous items are used with a single facet and with constraint=cases.

Note that if *`JML`* estimation is used then *`quick`* is the default and *`empirical`* is not available.

For pairwise models, the option `stderr` is not relevant and is ignored.

`distribution=`*`type`*

Specifies the (conditional) distribution that is used for the latent variable. *`type`* can take the value *`normal`*, or *`discrete`*. The default is *`normal`*. If *`discrete`* is chosen fit statistics cannot be computed. This option is not available with JML estimation. A *`discrete`* distribution is not available with regressors.

`fit=`*`reply`*

Generates item fit statistics that will be included in the tables created by the `show` statement. If *`reply`* is *`no`*, fit statistics will be omitted from the `show` statement tables. The default is *`yes`* (see also the `estimates` option of the `show` command).

`deviancechange=`*`f`*

Instructs estimation to terminate when the change in the deviance between successive iterations of the EM algorithm is less than *`f`*. The default value is *`0.0001`*.

`abilities=`*`reply`*

If *`reply`* is *`yes`*, ability estimates (WLE, MLE, EAP and plausible values) will be generated after the model has converged. This may accelerate later commands that require the use or display of these estimates. The default is *`no`*.

`history=`*`filename`*

When the `history` option is specified, a file containing all parameter estimates are each iteration named *`filename`* will be created. This is equivalent to using the `history` command after estimation. The default is to create no history file.

`ifit=`*`f`*

Same as the `fit` option.

`pfit=`*`f`*

Computes case fit estimates following estimation. They are then accessible on conjunction with the `matrixout` option.

`matrixout=`*`name`*

`name` is a matrix (or set of matrices) that will be created and will hold the results. These results are stored in the temporary workspace. Any existing matrices with matching names will be overwritten without warning. The contents of the matrices is described in the section ***Matrix Objects Created by Analysis Commands***.

`progress=`*`type`*

`type` is a colon (:) separated list of values for which a convergence progress plots is provided. Values permitted in the list are *`gins`*, *`covariance`*, *`regressors`*, *`scores`* and *`deviance`*.

Redirection

Redirection is not applicable to this command.

Examples

```
estimate;
```

Estimates the currently specified model using the default value for all options.

```
estimate ! method=jml;
```

Estimates the currently specified model using joint maximum likelihood.

```
estimate ! converge=0.0001, method=quadrature, nodes=15;
```

Estimates the currently defined model using the quadrature method of integration. It uses 15 nodes for each dimension and terminates when the change in parameter estimates is less than 0.0001 or after 200 iterations (the default for the `iterations` option), whichever comes first.

```
estimate ! method=montecarlo, nodes=200, converge=.01;
```

In this estimation, we are using the Monte Carlo integration method with 200 nodes and a convergence criterion of 0.01. This analysis (in conjunction with `export` statements for the estimated parameters) is undertaken to provide initial parameter estimates for a more accurate analysis that will follow.

```
estimate ! method=montecarlo, nodes=2000,
   plausible=mdim.pls;
```

Estimate the currently defined model using the Monte Carlo integration method with 2000 nodes. After the estimation, write plausible values and EAP estimates to the file `mdim.pls`.

```
score (0,1,2,3,4) (0,1,2,3,4) ( ) ! tasks(1-9);
score (0,1,2,3,4) ( ) (0,1,2,3,4) ! tasks(10-18);
model tasks + tasks*step;
estimate ! fit=no, method=montecarlo, nodes=400,
   converge=.01;
```

Initiates the estimation of a partial credit model using the Monte Carlo integration method to approximate multidimensional integrals. This estimation is done with 400 nodes, a value that will probably lead to good estimates of the item parameters, but the latent variance-covariance matrix may not be well estimated. Simulation studies suggest that 1000 to 2000 nodes may be needed for accurate estimation of the variance-covariance matrix. We are using 400 nodes here to obtain initial values for input into a second analysis that uses 2000 nodes. We have specified `fit=no` because we will not be generating any displays and thus have no need for this data at this time. We are also using a convergence criterion of just 0.01, which is appropriate for the first stage of a two-stage estimation.

GUI Access

Analysis ➔ Estimate.

Notes

(1) ConQuest offers three approximation methods for computing the integrals that must be computed in marginal maximum likelihood estimation (MML): quadrature (Bock/Aitken quadrature), gauss (Gauss-Hermite quadrature) and Monte Carlo. The gauss method is generally the preferred approach for problems of three or fewer dimensions, while the Monte Carlo method is preferred in problems with higher dimensions. Gauss cannot however be used when there are regressors or if the distribution is discrete.

(2) In the absence of regression variables, the *gauss* method is the default method. In the presence of regression variables *quadrature* is the default.

(3) Joint maximum likelihood (JML) cannot be used if any cases have missing data for all of the items on a dimension.

(4) The order in which command statements can be entered into ConQuest is not fixed. There are, however, logical constraints on the ordering. For example, `show` statements cannot precede the `estimate` statement, which in turn cannot precede the `model`, `format` or `datafile` statements, all three of which must be provided before estimation can take place.

(5) The iterations will terminate at the first satisfaction of any of the `converge, deviancechange and iterations` options.

(6) Fit statistics can be used to suggest alternative models that might be fit to the data. Omitting fit statistics will reduce computing time.

(7) Simulation results illustrate that 10 nodes per dimension will normally be sufficient for accurate estimation with the quadrature method.

(8) The `stderr=quick` is much faster than `stderr=empirical` and can be used for single faceted models with `constraint=cases`. In general, however, to obtain accurate estimates of the errors (for example, to judge whether DIF is observed by comparing the estimates of some parameters to their standard errors, or when you have a large number of facets, each of which has only a couple of levels) `stderr=quick` is not advised.

(9) It is possible to recover the ConQuest estimate of the latent ability correlation from the output of a multidimensional analysis by using plausible values. Plausible values can be produced through the `estimate` command or through the `show` command with argument `cases` in conjunction with the option `estimates=latent`.

(10) The default settings of the `estimate` command will result in a Gauss-Hermite method that uses 15 nodes for each latent dimension when performing the integrations that are necessary in the estimation algorithm. For a two-dimensional model, this means a total of 15×15=225 nodes. The total number of nodes that will be used increases exponentially with the number of dimensions, and the amount of time taken per iteration increases linearly with the number of nodes. In practice, we have found that a total of 4000 nodes is a reasonable upper limit on the number of nodes that can be used.

(11) If the estimation method chosen is JML, then it is not possible to estimate item scores.

(12) In the case of MML estimation, ability estimate matrices are only available if `abilities=yes`, is used.

execute

Runs all commands up to the execute command.

Argument

This command does not have an argument.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
let length=50;
execute;
dofor i=1-1000;
data file_%i%.dat;
format responses 1-%length%;
model item;
estimate;
show >> results_%i%.shw;
enddo;
```

If this code is submitted as a batch, the `execute` command ensures the length token is substituted prior to the execution of the loop. Without the `execute` the substitution of the token would occur after the loop is executed, which would result in much slower command parsing.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) The `execute` command is the only ConQuest command that cannot be abbreviated and must be in lower case and with no space prior to the semi-colon.

(2) An `execute` statement cannot be contained in a loop.

export

Creates files that contain estimated values for any of the parameters, a file that contains the design matrix used in the estimation, a scored data set, an iteration history, or a log file containing information about the estimation.

Argument

`info type`

`info type` takes one of the values in the following list and indicates the type of information that is to be exported. The format of the file that is being exported will depend upon the *`info type`*.

`parameters` or `xsi`

The file will contain the estimates of the item response model parameters. If text output is requested the format of the file is identical to that described for the `import` command argument `init_parameters`.

`reg_coefficients` or `beta`

The file will contain the estimates of the regression coefficients for the population model. If text output is requested the format of the file is identical to that described for the `import` command argument `init_reg_coefficients`.

`covariance` or `sigma`

The file will contain the estimate of the variance-covariance matrix for the population model. If text output is requested the format of the file is identical to that described for the `import` command argument `init_covariance`.

`tau`

The file will contain the estimates of the item scoring parameters. If text output is requested the format of the file is identical to that described for the `import` command argument `init_tau`.

`itemscores`

The file will contain the estimated scores for each category of each item on each dimension.

`designmatrix` or `amatrix`

The file will contain the design matrix that was used in the item location parameter estimation. The format of the file will be the same as the format required for importing a design matrix.

`cmatrix`

The file will contain the design matrix that was used in the scoring parameter estimation. The format of the file will be the same as the format required for importing a design matrix.

`logfile`

The file will contain a record of all statements that are issued after it is requested, and it will contain results on the progress of the estimation.

`scoreddata`

The file will contain scored item response vectors for each case. The file contains one record per case. It includes a sequence number, then a pid (if provided) followed by scored responses to each (generalised) item.

`history`

The file will contain a record for each estimation iteration showing the deviance and parameter estimates at that time.

Options

`filetype=`*`type`*

`type` can take the value *`spss`*, *`excel`* or *`text`*. It sets the format of the output file. This option does not apply to the argument `logfile` or `history`. The default is *`text`*.

Redirection

`>>` *`filename`*

An export file name must be specified.

Examples

```
export parameters >> p.dat;
```

Item response model parameters are to be written to the file `p.dat`.

```
export parameters >> p.dat, reg_coefficients >> r.dat;
```

Multiple exports can be specified with one statement.

GUI Access

File ➔ Export.

Export of each of the file types is accessible as a file menu item.

Notes

(1) If using text output the format of the export files created by the `xsi`, `beta`, `sigma` and `tau` arguments matches the format of ConQuest import files so that export files can be re-read as either anchor files or initial value files. See the `import` command for the formats of the files.

(2) The `logfile` argument can be used at any time. The `scoreddata`, `itemscores`, `history`, `amatrix` and `cmatrix` arguments are only available after a model has been estimated. The other arguments are only possible after a model has been estimated. The `xsi`, `tau`, `beta`, and `sigma` arguments can be used prior to estimation but only

in conjunction with text output. In this case, the files are updated after each iteration.

(3) The export file names remain specified until a `reset` statement is issued. This means that, if export file names are not changed between `estimate` statements, an attempt will be made to write new data to an existing file. This will result in a loss of the previous data.

(4) The best strategy for manually building a design matrix (either item location or scoring) usually involves running ConQuest, using a `model` statement to generate a design matrix, and then exporting the automatically generated matrix, using the `amatrix` and `cmatrix` arguments. The exported matrix can then be edited as needed and then imported.

filter

Allows specification of a set of item-case combinations that can be omitted from the analysis. Filtering can be based on data in a file or in a matrix variable. The file (or matrix variable) can contain '0' or '1' filter indicators or real values tested against a specified value.

Argument

This command does not have an argument.

Options

`method=`*`reply`*

`reply` takes the value *`binary`*, *`value`* *or* *`range`*. If *`binary`* is used then it is assumed that input data consists of zeros and ones, and item case combinations with a value of '1' are retained. Those with the value '0' will be filtered out of subsequent analyses. If *`reply=value`* then the value is tested against the *`match`* option. If *`reply=range`* the value is tested against the *`min`* and *`max`* options. The default is *`value`*.

`matrixin=`*`name`*

`name` is a matrix variable used as the data source. The dimensions must be number of cases by number of items. This option cannot be used in conjunction with an infile redirection.

`matrixout=`*`name`*

`name` is a matrix variable that is created and with dimensions number of cases by number of items. It will contain a value of '1' for case item combinations retained and a value of '0' for those case item combinations that are filtered out of subsequent analyses.

`filetypein=`*`type`*

`type` can take the value *`spss`* or *`text`*. This option describes the format of infile. If an SPSS file is used, it must have the same number of cases as the data set that is being analysed and it must have number of items plus 2 variables. The first two variables are ignored and the remaining variables provide data for each item. When used with *`method`* or with *`min`* and *`max`* options, the variables must be numeric. The default is *`text`*.

`filetypeout=`*`type`*

`type` can take the value *`spss, excel, xls, xlsx`* or *`text`*. This option sets the format of the results file. The default is text.

`match=`*`value`*

Case/item combinations for which the input data matches *`value`* are omitted from analysis, whilst those that do not match are retained. Requires the *`method=value`* option.

`min=`*`n`*

Case/item combinations for which the input data are less than *`n`* are omitted from analysis. Requires the *`method=range`* option. The default is *`0`*.

`max=`*`n`*

Case/item combinations for which the input data are greater than *`n`* are omitted from analysis. Requires the *`method=range`* option. The default is *`1`*.

Redirection

`<< infilename`

Name of file to read filtering data from.

`>>` *`outfilename`*

A file name for a file of ones and zeros showing which cases/item combinations are retained or omitted.

Examples

```
filter ! filetypein=spss, method=value, match=T
   << filter.sav;
```

Filters data when a value of "*`T`*" is provided for the case item combinations in the SPSS system file `filter.sav`.

```
filter ! matrixin=f, method=range, min=0.25;
```

Filters data when the value in *`f`* associated with a case/item combination is less than or equal to `0.25`, or greater than or equal to 1.0

GUI access

Access to this command through the GUI is not available.

Notes

(1) The most common utilisation of filter is to remove outlying observations from the analysis.

(2) The format of the SPSS system file produced by *`show expected`* matches that required by filter as SPSS input file.

(3) Filtering is turned on by the filter command and stays in place until a reset command is issued.

fit

Produces residual-based fit statistics.

Argument

The optional argument takes the form *L*1:*L*2:…:*LN* Where *Lk* is a list of column numbers in the default fit design matrix. This results in *N* fit tests. In the fit tests the columns in each list are summed to produce a new fit design matrix.

Either an argument or an input file can be specified, but not both.

Options

`groups=`*`variable`*

An explicit variable. Fit statistics will be provided for each level of the grouping variable. The *`variable`* must have been listed in a previous `group` command.

`matrixout=`*`name`*

`name` is a matrix (or a set of matrices) that will be created and will hold the fit results. The matrix will be added to the workspace. Any existing matrices with matching names will be overwritten without warning. The contents of the matrices is described in the section **Matrix Objects Created by Analysis Commands.**

`filetype=`*`type`*

`type` can take the value *`spss`*, *`excel`*, *`xls`*, *`xlsx`* or *`text`*. This option sets the format of the output file. The default is *`text`*.

Redirection

`<<` *`infilename`*

A file name for the fit design matrix can be specified. The fit design matrix has the same format as a model design matrix (see `import designmatrix`).

`>>` *`outfilename`*

A file name for the output of results.

Examples

```
fit >> fit.res;
```

Use the default fit design matrix and writes results to the file *`fit.res`*.

```
fit 1-3:4,5,7 >> fit.res;
```

Performs two fit tests. The first test is based upon the sum of the first three columns of the default fit design matrix and the second is based upon the sum of columns, 4, 5 and 7 of the default fit design matrix. Results are written to the file *`fit.res`*.

```
fit << fit.des >> fit.res;
```

Uses the fit design matrix in the file `fit.des` and write results to the file `fit.res`.

GUI Access

Analysis ➔ Fit.

Selecting the fit menu item displays the open file dialog box for selection of a file that contains the fit design matrix.

Notes

(1) An argument and `infile` cannot be combined.

(2) At the moment, `filetype=`*`excel`* is the same as `filetype=`*`xls`* or `filetype=`*`xlsx`*.

for

Allows looping of syntax and loop control for the purposes of computation.

Argument

```
(range) {
   set of ConQuest commands
   };
```

`range` is an expression that must take the form *var in low:high* where *var* is a variable and *low* and *high* evaluate to integer numeric values. The numeric values can be a scalar value, a reference to an existing 1x1 matrix variable or a 1x1 submatrix of an existing matrix variable. The numeric values cannot involve computation.

The set of commands is executed with *var* taking the value low through to high in increments of one.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
let x=matrix(6:6);
compute k=1;
for (i in 1:6)
    {
      for (j in 1:i)
      {
       compute x[i,j]=k;
       compute k=k+1;
      };
    };
print x; print ! filetype=xlsx >> x.xlsx;
print ! filetype=spss >> x.sav;
```

Creates a 6 by 6 matrix of zero values and then fills the lower triangle of the matrix with the numbers 1 to 21. The matrix is then printed to the screen and saved as both an Excel and an SPSS file.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) There are no limits of the nesting of loops.

format

Describes the layout of the data in a data file by specifying variable names and their locations (either explicitly by column number or implicitly by the column locations that underlie the `responses` variable) within the data file.

Argument

A list of space-delimited variables that are to be analysed. Each variable is followed by a column specification.

Every `format` statement argument must include the reserved variable `responses`. The `responses` variable specifies the location of the 'item' responses. The column specifications for `responses` are referred to as the *response block*.

A response-width indicator can be given after the final response block. The width indicator, `(a`*`n`*`)`, indicates that the width of each response is *`n`* columns. All responses must be of the same width.

The reserved variable `pid` links data that are from a single case but are located in different records in the data file. It provides a case identification variable that will be included in case outputs. By default `pid` links data that are from a single case but are located in different records in the input data file. See notes (2), (3) and (14), and the `set` option `uniquepid`.

Additional user-defined variables that are listed in the argument of a format statement are called *explicit variables*.

The reserved word `to` can be used to indicate a range of variables.

A slash (`/`) in the `format` statement argument means move to the next line of the datafile (see note (5)).

Options

A list of user-provided, comma-separated variables that are implicitly defined through the column locations that underlie the `responses` variable. The default implicit variable is *`item`* or *`items`*, and you may use either in ConQuest statements.

Redirection

Redirection is not applicable to this command.

Examples

```
format class 2 responses 10-30 rater 43-45;
```

The user-defined explicit variable `class` is in column 2. Item 1 of the response data is in column 10, item 2 in column 11, etc. The user-defined explicit variable `rater` is in columns 43 through 45.

```
format responses 1-10,15-25;
```

The response data are not stored in a contiguous block, so we have used a comma (`,`) to separate the two column ranges that form the response block. The above example states that response data are in columns 1 through 10 and columns 15 through 25. Commas are not allowed between explicit variables or within the column specifications for other variables.

```
format responses 1-10 / 1-10;
```

Each record consists of two lines. Columns 1 through 10 on the first line of each record contain the first 10 responses. Columns 1 through 10 on the second line of each record contain responses 11 through 20.

```
format responses 21-30 (a2);
```

If each response takes more than one column, use `(a`*`n`*`)` (where *`n`* is an integer) to specify the width of each response. In the above example, there are five items. Item 1 is in columns 21 and 22, item 2 is in columns 23 and 24, etc. All responses must have the same width.

```
format class 3-6 rater 10-11 responses 21-30 rater 45-46
    responses 51-60;
```

Note that `rater` occurs twice and that `responses` also occurs twice. In this data file, two raters gave ratings to 10 items. The first rater's identifier is in columns 10 and 11, and the corresponding ratings are in columns 21 through 30. The second rater's identifier is in columns 45 and 46, and the corresponding ratings are in columns 51 through 60. There is only one occurrence of the variable `class` (in columns 3 through 6). This variable is therefore associated with both occurrences of `responses`. If explicit variables are repeated in a `format` statement, the *n*-th occurrence of `responses` will be associated with the *n*-th occurrence of the other variable(s); or if *n* is greater than the number of occurrences of the other variable(s), the *n*-th occurrence of `responses` will be associated with the highest occurrence of the other variable(s).

```
format responses 11-20 ! task(10);
```

The option `task(10)` indicates that we want to refer to the implicit variable that underlies `responses` as 10 tasks. When no option is provided, the default name for the implicit variable is *`item`*.

```
format responses 11-20 ! item(5), rater(2);
```

The above example has two user-defined implicit variables: `item` and `rater`. There are five items and two raters. Columns 11 through 15 contain the ratings for items 1 through 5 by rater 1. Columns 16 through 20 contain the ratings for items 1 through 5 by rater 2. In general, the combinations of implicit variables are ordered with the elements of the leftmost variables cycling fastest.

```
format responses 1-48 ! criterion(8), essay(3), rater(2);
```

Columns 1 through 8 contain the eight ratings on essay 1 by rater 1, columns 9 through 16 contain the eight ratings on essay 2 by rater 1, and columns 17 through 24 contain the eight ratings on essay 3 by rater 1. Columns 25 through 48 contain the ratings by rater 2 in a similar way.

```
format pid 1-5 class 12-14 responses 31-50 rater 52-53;
```

The identification variable `pid` is in columns 1 through 5. The variable `class` is in columns 12 through 14. Item response data are in columns 31 through 50. The `rater` identifier is in columns 52 and 53. Here we have assumed that a number of raters have rated the work of each student and that the ratings of each rater have been entered in separate records in the data file. The specification of the `pid` will ensure that all of the records of a particular case are located and identified as belonging together.

```
format pid 1-5 var001 to var100 100-199;
```

The identification variable `pid` is in columns 1 through 5. A set of explicit variables labelled var01 through var100 are defined and read from columns 100-199.

GUI Access

Command ➔ Format.

This dialog box can be used to build a format command. Selecting each of the radio buttons in turn allows the specification of explicit variables, responses and implicit variables. Each specification needs to be added to the format statement.

Notes

(1) User-provided variable names must begin with an alphabetic character and must be made up of alphabetic characters or digits. Spaces are not allowed in variable names. A number of reserved words that cannot be used as variable names are provided in the ***List of illegal characters and words for variable names,*** at the end of this document.

(2) The reserved explicit variable `pid` means person identifier or case identifier. If `pid` is not specified in the `format` statement, then ConQuest generates identifier values for each record on the assumption that the data file is 'by case'. If `pid` is specified, ConQuest sorts the records in order of the `pid` field first before processing. While this means that the data for each case need not be all together and thus allows for flexibility in input format, the cost is longer processing time for doing the sort..

(3) If `pid` is specified, output to person estimates files include the `pid` and will be in `pid` order. Otherwise output to the files will be in sequential order.

(4) The `format` statement is limited to reading 50 lines of data at a time. In other words, the maximum number of slash characters you can use in a `format` statement is 49. See note (8) for the length of a line.

(5) The total number of lines in the data set must be exactly divisible by the number of lines that are specified by the use of the slash (`/`) character in the `format` statement. In other words, each record must have the same number of lines.

(6) Commas can only be used in the column specifications of the `responses` variable. Column specifications for all other explicit variables must be contiguous blocks.

(7) The width (number of columns) specified for each response variable must be the same. For example, the following is ***not*** permitted:

```
format responses 1-4 (a2) responses 5-8 (a1);.
```

(8) The maximum number of columns in a data file must be less than 3072.

(9) If the `format` statement does not contain a `responses` variable in its argument, ConQuest will display an error message.

(10) In Rasch modelling, it is usual to identify the model by setting the mean of the item difficulty parameters to zero. This is also the default behaviour for ConQuest, which automatically sets the value of the 'last' item parameter to ensure an average of zero. If you want to use a different item as the constraining item, then you can read the items in a different order. For example:

```
format id 1-5 responses 12-15, 17-23, 16;
```

(11) would result in the constraint being applied to the item in column 16. But be aware, it will now be called item 12, not item 5, as it is the twelfth item in the response block.

(12) The level numbers of the `item` variable (that is, item 1, item 2, etc.) are determined by the order in which the column locations are set out in the response block. If you use

```
format responses 12-23;
```

item 1 will be read from column 12.

If you use

```
format responses 23,12-22;
```

item 1 will be read from column 23.

(13) In some testing contexts, it may be more informative to refer to the `responses` variable as something other than `item`. Specifying a user-defined variable name, such as `task` or `question`, may lead to output that is better documented. However, the new variable name for `responses` must then be used in the `model`, `labels`, `recode`, and `score` statements and any label file to indicate the `responses` variable.

(14) If each case has a unique `pid` and the data file contains a single record for each case then use of the `set` option `uniquepid=yes` will result in the `pid` being included in case output files, but processing speed will be increased. This is particularly useful for large data sets (e.g., greater than 10 000 cases) with unique student identifiers. This option should not be used without prior confirmation that the identifiers are unique.

generate

Generates data files according to specified options. This can be used to generate a single data set.

Argument

This command does not have an argument.

Options

`nitems=`*`n1:n2:…:nd`*

`ni` is the number of items on *i*-th dimension and *`d`* is the number of dimensions. The default is one dimension of 50 items.

`npersons=`*`p`*

`p` is the number of people in the test. The default is *`500`*.

`maxscat=`*`k`*

`k` is the maximum number of scoring categories for each item. For example, if the items are dichotomous, *`k`* should be 2. Note that *`k`* applies to all items, so you can't generate items with different numbers of categories. The default value is *`2`*.

`itemdist=`*`type`*

`type` is one of three arguments for specifying the item difficulties distribution: `normal(`$\boldsymbol{\mu}$`:`*`b`*`)`, `uniform(`*`c`*`:`*`d`*`)`, or `file`. `normal(`$\boldsymbol{\mu}$`:`*`b`*`)` draws item difficulties from a normal distribution with mean $\boldsymbol{\mu}$ and variance *b*. `uniform(`*`c`*`:`*`d`*`)` draws item difficulties from a uniform distribution with range *c* to *d*. `file` allows you to supply the item difficulties in a file by giving the file name. The file should be a standard text file with one line per item parameter. Each line should indicate, in the order given, the item number, the step number and the item parameter value.

For example, the file might look like:

```
1 0 -2.0
1 1  0.2
1 2  0.4
2 0 -1.5
..................
```

Note that the lines with a step number equal to 0 give the item difficulty and that the lines with a step number greater than 0 give the step parameters.

The default value is *`uniform(-2:2)`*.

`centre=`*`reply`*

Sets the location of the origin for the generated data. If *`reply`* is *`cases`*, the items parameters are left as randomly generated, and the cases are adjusted to have a mean of zero. If *`reply`* is *`items`*, the item location parameters are set to a mean of zero and the cases are left as generated. If `reply` is `no`, both cases and items are left as generated. The default is `items`.

`scoredist=type`

`type` is one of three arguments for specifying the item scores (ie discrimination) distribution: `normal(μ:b)`, `uniform(c:d)`, or `file`. `normal(μ:b)` draws item scores from a normal distribution with mean *μ* and variance *b*. `uniform(c:d)` draws item scores from a uniform distribution with range *c* to *d*. `file` allows you to supply the item scores in a file by giving the file name. The file should be a standard text file with one line per item parameter. Each line should indicate, in the order given, the item number, the step number and the item score value.

For example, the file might look like:

```
1 1 1.0
1 2 1.5
2 1 0.8
..................
```

The default value is for the scores to be set equal the category label. That is for the Rasch model to apply.

`abilitydist=type`

`type` is one of eight arguments for specifying the distribution of the latent abilities:

```
normal(μ:b)
normal2(μ1:b1:μ2:b2:k)
normalmix(μ1:b1:μ2:b2:p)
uniform(c:d)
t(d)
chisq(d)
mvnormal(μ1:b1:μ2:b2:…:μd:bd:r12…r(d-1)(d-1))
file
```

`normal(μ:b) draws` abilities from a normal distribution with mean *μ* and variance *b*. `normal2(μ1:b1:μ2:b2:k)` draws abilities from a two-level normal distribution. Students are clustered in groups of size *k*. The within group mean and variance are ***μ1*** and *b1* respectively, while the between group mean and variance are ***μ2*** and *b2* respectively. If a two-level distribution is specified the group-level means of the generated values are written to the generated data file for use in subsequent analysis. `normalmix(μ1:b1: μ2:b2:p)` draws abilities from a mixture of two normal distributions with group one mean and variance ***μ1*** and *b1*, and group two mean and variance ***μ2*** and *b2*. *p* is the proportion of the mixture that is sampled from group one. `uniform(c:d)` draws abilities from a uniform distribution with range *c* to *d*. `t(d)` draws abilities from a t distribution with *d* degrees of freedom. `chisq(d)` draws abilities from a standardised (ie scaled to mean zero and standard deviation one) chi squared distribution with *d* degrees of freedom. `mvnormal(μ1:b1:μ2:b2:…:μd:bd:r12:…:r1d:r23:…:r(d-1)(d))` draws abilities from a *d*-dimensional multivariate normal distribution. ***μ1*** to ***μd*** are the means for each of the dimensions, *b1* to *bd* are the variances and *r12* to *r(d-1)(d-1)* are the correlations between the dimensions. For example, a 3-dimensional multivariate distribution with the following mean vector and variance matrix:

$$\begin{pmatrix} 0.5 \\ 1.0 \\ 0.0 \end{pmatrix} \qquad \begin{pmatrix} 1.0 & 0 & -0.2 \\ 0 & 1.0 & 0.8 \\ -0.2 & 0.8 & 1.0 \end{pmatrix}$$

is specified as `mvnormal(0.5:1:1:1:0:1:0:-0.2:0.8)`

file allows you to supply the abilities in a file by giving the file name. If the option `importnpvs` is NOT being used the file should be a standard text file with one line per case. Each line should indicate, in the order given, the case number, and the ability value.

For example, the file might look like:

```
1 -1.0
1  0.23
1 -0.45
2 -1.5
```

If the option `importnpvs` is being used then the file format should match that of a file produced by `show cases ! estimates=latent`. The number of plausible values and dimensions in the file must match the numbers specified by `importnpvs` and `importdims`.

The default value is *normal(0:1)*.

`regfile=`*filename(v1:v2:v3:…:vn)*

filename is a file from which a set of regression variables can be read. The names of the regression variables are given in parenthesis after the file name, an separated by colons (:) *v1:…:vn*.

The values of the regression variables are written into the generated data file for use in subsequent analysis.

The first line of the file must give *n* regression coefficients. This is followed by one line per person. Each line should indicate, in the given order, the case number and then the value or regression variable *v1*, then *v2*, and so on, until *vn*.

For example, the file might look like:

```
  3.0 2.1 -0.5
1 0.230 0.400 -3.000
2 -0.450 0.500 2.000
3 -1.500 3.222 -4.000
```

`model=`*model name*

Set the type of model. The only valid model name is *pairwise* which results in the generation of data that follows the Bradley–Terry–Luce (BTL) model.

`matrixout=`*`name`*

`name` is a matrix (or set of matrices) that will be created and will hold the results. Any existing matrices with matching names will be overwritten without warning. The content of each of the matrices is described in the section ***Matrix Objects Created by Analysis Commands***.

`importnpvs=`*`n`*

`n` is the number of plausible values in an import file that will be used to produce multiple output data sets, one for each plausible value set

`importdims=`*`n`*

`n` is the number of dimensions in an import file that will be used to produce multiple output data sets, one for each plausible value set

`group=`*`variable`*

An explicit variable to be used as grouping variable. Used only when importing plausible value and undertaking a posterior predictive model checking. If a group is specified then summary statistics are saved as matrix variables for each group. Groups can only be used if they have been previously defined by a group command and a model has been estimated.

Redirection

`>>` *`filename1, filename2, filename3`*

`filename1` is the name of the generated data file. *`filename2`* and *`filename3`* are optional. *`filename2`* is the name of the generated item difficulties file, and *`filename3`* is the name of the generated abilities file. When *`abilitydist=normal2`* is used the mean of each groups abilities is also written to *`filename1`*. The mean is for all students in the group with the current student excluded. When `regfile=`*`filename`* is used the regression variables are also written to *`filename1`*. If the `scoredist` argument is used and a `filename2` is requested then an additional file with the name `filename2_scr` is created and it contains the generated score parameters.

When the option `importnpvs` is used then a set of data files with names `filename1_pv`*`n`*`.dat` will be produced, where *n* runs from one to the number of plausible values

Examples

```
generate ! nitems=30, npersons=300, maxscat=2,
   itemdist=item1.dat, abilitydist=normal(0:1) >>
   sim1.dat;
```

A data set called `sim1.dat` is created. It contains the responses of 300 students to 30 dichotomously scored items. The generating values of the item difficulty parameters are read from the file `item1.dat`, and the latent abilities for each person are randomly drawn from a unit normal distribution with zero mean and a variance of 1.

```
generate ! nitems=20, npersons=500, maxscat=3,
    itemdist=uniform(-2:2), abilitydist=normal(0:1.5)
    >> sim1.dat, sim1.itm, sim1.abl;
```

A data set called `sim1.dat` is created along with a file containing the generating values of the item parameters (`sim1.itm`) and another containing the generating values of the latent abilities (`sim1.abl`). The data set will contain the generated responses of 500 persons to 20 partial credit items with three response categories that are scored 0, 1 and 2 respectively. All of the item parameters were randomly drawn from a uniform distribution with minimum -2 and maximum 2, and the abilities are drawn from a normal distribution with zero mean and a variance of 1.5.

```
generate ! nitems=20, npersons=500,
    maxscat=3,scoredist=uniform(0.5:2),
    itemdist=uniform(-2:2), abilitydist=normal(0:1.5)
    >> sim1.dat, sim1.itm, sim1.abl;
```

As for the previous example but with scoring parameters generated and written to the file sim1_scr.itm.

```
generate ! nitems=20, npersons=500, maxscat=3,
    abilitydist=normal2(0:0.7:0:0.3:20)
    >> sim1.dat, sim1.itm, sim1.abl;
```

A data set called `sim1.dat` is created along with a file containing the generating values of the item parameters (`sim1.itm`) and another containing the generating values of the latent abilities (`sim1.abl`). The data set will contain the generated responses of 500 persons to 20 partial credit items with three response categories that are scored 0, 1 and 2 respectively. All of the item parameters were randomly drawn from a uniform distribution with minimum -2 and maximum 2 (default). The abilities are drawn from a two-level normal distribution with within group zero mean and a variance of 0.7, and between group zero mean and variance of 0.3. The group size is 20. The means of the generated abilities for each group will also be written to the data set (`sim1.dat`). Note that the group mean excludes the current student.

```
generate ! nitems=30, npersons=300, maxscat=2,
    itemdist=item1.dat, abilitydist=normal(0:1),
    regfile=reg1.dat(gender:ses)>> sim1.dat;
```

A data set called `sim1.dat` is created. It contains the responses of 300 students to 30 dichotomously scored items. The generating values of the item difficulty parameters are read from the file `item1.dat`, and the latent abilities for each person are randomly drawn from the regression model $\theta = \alpha_1 gender + \alpha_2 ses + \varepsilon$ where $\alpha_1 gender + \alpha_2 ses$ is computed based upon the information given in `reg1.dat` and ε is randomly generated as a unit normal deviate with zero mean and a variance of 1.

```
generate ! nitems=30, npersons=3000, maxscat=2,
    scoredist=uniform(0.5:2), abilitydist=normal(0:1),
    matrixout=2pl >> sim1.dat;
```

A data set called `sim1.dat` is created. It contains the responses of 3000 students to 30 dichotomously scored items with scoring parameters randomly drawn from a uniform distribution with minim 0.5 and maximum 2. The generating values of the item difficulty parameters use the default of a uniform distribution with minimum -2 and maximum 2, and the latent abilities for each person are randomly drawn from a unit normal distribution. The matrixout results in the production of four matrix variables 2pl_items, 2pl_cases, 2pl_scores and 2pl_responses.

```
generate ! nitems=15:15, npersons=3000, maxscat=2,
    scoredist=uniform(0.5:2),
    abilitydist=mvnormal(0:1:0:1:0.5), matrixout=2d2pl >>
    sim1.dat;
```

A data set called `sim1.dat` is created. It contains the responses of 3000 students to 30 dichotomously scored items, 15 for each of two dimensions. Scoring parameters are randomly drawn from a uniform distribution with minim 0.5 and maximum 2. The generating values of the item difficulty parameters use the default of a uniform distribution with minimum -2 and maximum 2. The latent abilities for each person are randomly drawn from a bivariate standard normal distribution with correlation 0.5. The `matrixout` option results in the production of four matrix variables 2d2pl_items, 2d2pl_cases, 2d2pl_scores and 2d2pl_responses.

```
generate ! nitems=15:15,importnpms=50,importdim=2,
    npersons=3000,scoredist=uniform(0.5:2),
    abilitydist=ex1.pv, matrixout=ex1 >> sim1.dat;
```

A set of data sets called `sim1_pv1.dat` to `sim1_pv50.dat` are created. The contains the responses of 3000 students to 30 dichotomously scored items, 15 for each of two dimensions based upon the plausible values provided in ex1.pv. Scoring parameters are randomly drawn from a uniform distribution with minim 0.5 and maximum 2. The generating values of the item difficulty parameters use the default of a standard normal distribution The `matrixout` option results in the production of four matrix variables ex1 _items, ex1 _scores and ex1 _statistics.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) The `generate` command is provided so that users interested in simulation studies can easily create data sets with known characteristics.

(2) If *`abilitydist=normal2(μ1:b1:μ1:b1:k`*) is used, the total number of persons must be divisible by k.

(3) The random number generation is seeded with a default value of "*1*". This default can be changed with the *seed* option in the *set* command. Multiple runs of *generate* within one session use a single random number sequence, so any change to the default seed should be made before the first generate command is issued.

(4) The *pairwise* model is undimensional and does not use discrimination or ability parameters.

get

Reads a previously saved system file.

Argument

This command does not have an argument.

Options

This command does not have any options.

Redirection

```
<< mysysfile.sys;
```

`mysysfile.sys` is the name of a ConQuest system file saved during a previous ConQuest session or earlier in the current session.

Example

```
get << mysysfile.sys;
```

Loads the system file `mysysfile.sys`.

GUI Access

File ➔ Get System File.

Notes

(1) Loading a system file replaces all previously entered commands.

group

Specifies the grouping variables that can be used to subset the data for certain analyses and displays.

Argument

A list of explicit variables to be used as grouping variables. The list can be comma-delimited or space-delimited.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
group age grade gender;
```

Specifies `age`, `grade` and `gender` as grouping variables.

GUI Access

Command ➔ Grouping Variables.

The available grouping variables are shown in the list. Multiple groups can be selected by shift- or control-clicking.

Notes

(1) Each of the grouping variables that are specified in a `group` statement must take only one value for each measured object (typically a person), as these are 'attribute' variables for each person. For example, it would be fine to use *`age`* as a group variable, but it would not make sense to use *`item`* as a regression variable.

(2) Group variables are read as strings. If using group variables read from SPSS files that are Numeric in type, they will be converted to strings. See Note 5 in `datafile.`

(3) The *`group`* statement stays in effect until it is replaced with another *`group`* statement or until a *`reset`* statement is issued.

(4) The *`group`* statement must be specified prior to estimation of the model.

if

Allows conditional execution of commands.

Argument

```
(logical condition) {
   set of ConQuest commands
   };
```

If logical condition evaluates to `true`, the set of ConQuest commands is executed. The commands are not executed if the logical condition does not evaluate to `true`.

The logical condition can be `true`, `false` or of the form *s*1 operator *s*2, where *s*1 and *s*2 are strings and operator is one of the following

```
==   equality
=>   greater than or equal to
>=   greater than or equal to
=<   less than or equal to
<=   less than or equal to
!=   not equal to
>    greater than
<    less than
```

For each of *s*1 and *s*2 ConQuest first attempts to convert it to a numeric value. The numeric value can be a scalar value, a reference to an existing 1x1 matrix variable or a 1x1 submatrix of an existing matrix variable. A numeric value cannot involve computation.

If *s*1 is a numeric value the operator is applied numerically. If not a string comparison occurs between *s*1 and *s*2.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
let x=matrix(20:20);
compute k=1;
for (i in 1:20)
    {
      for (j in 1:i)
      {
       if (j<i)
       {
         compute x[i,j]=k;
         compute x[j,i]=-k;
         compute k=k+1;
       };

       if (j==i)
       {
        compute x[i,j]=j;
       };
      };
    };

print x;
```

Creates a 20 by 20 matrix of zero values and then fills the lower triangle of the matrix with the numbers 1 to 190, the upper triangle with -1 to -190 and the diagonal with the numbers 1 to 20. The matrix is then printed to the screen.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) There are no limits on the nesting of conditions.

import

Identifies files that contain initial values for any of the parameter estimates, files that contain anchor values for any of the parameters, or a file that contains a design matrix.

Argument

`info type`

`info type` takes one of the values in the following list and indicates the type of information that is to be imported. The format of the file that is being imported will depend upon the *`info type`*.

`init_parameters or init_xsi`

Indicates initial values for the response model parameters. The file will contain two pieces of information for each response model parameter that has an initial value specified: the parameter number and the value to use as the initial value. The file must contain a sequence of values with the following pattern, in the order given: parameter number, initial value, parameter number, initial value, and so forth.

For example, the following may be the contents of an `init_parameters` file:

```
1   0.567
2   1.293
3   -2.44
8   1.459
```

`init_tau`

Indicates initial values for the tau scoring parameters used with the *`scoresfree`* option. The file will contain two pieces of information for each tau parameter that has an initial value specified: the parameter number and the value to use as the initial value. The file must contain a sequence of values with the following pattern, in the order given: parameter number, initial value, parameter number, initial value, and so forth. Details of the tau parameterisation can be found in ConQuest note "Score Estimation and Generalised Partial Credit Models (revised)".

For example, the following may be the contents of an `init_tau` file:

```
1   0.5
2   1.293
3   2.44
4   1.459
```

`init_reg_coefficients or init_beta`

Indicates initial values for the regression coefficients in the population model. The file will contain three pieces of information for each regression coefficient that has an initial value specified: the dimension number, the regression coefficient number, and the value to use as the initial value. Dimension numbers are integers that run from 1 to the number of dimensions, and regression coefficient numbers are integers that run from 0 to the number of regressors. The zero is used for the constant term. When there are no regressors, 0 is the mean. The file must contain a sequence of values with the following pattern: dimension number, regressor number, initial value, dimension number, regressor number, initial value, and so forth.

For example, the following may be the contents of an `init_reg_coefficients` file:

```
1   0  0.233
2   0  1.114
1   1 -0.44
2   1 -2.591
```

If you are fitting a one-dimensional model, you must still enter the dimension number. It will, of course, be 1.

`init_covariance or init_sigma`

Indicates initial values for the elements of the population model's variance-covariance matrix. The file will contain three pieces of information for each element of the covariance matrix that has an initial value specified: the two dimension specifiers and the value to use as the initial value. Dimension specifiers are integers that run from 1 to the number of dimensions. As the covariance matrix is symmetric, you only have to input elements from the upper half of the matrix. In fact, ConQuest will only accept initial values in which the second dimension specifier is greater than or equal to the first. The file must contain a sequence of values with the following pattern: dimension specifier one, dimension specifier two, initial value, dimension specifier one, dimension specifier two, initial value, and so forth.

For example, the following may be the contents of an `init_covariance` file

```
1   1  1.33
1   2 -0.11
2   2  0.67
```

If you are fitting a one-dimensional model, the variance-covariance matrix will have only one element: the variance. In this case, you must still enter the dimension specifiers in the file to be imported. They will, of course, both be 1.

`anchor_parameters or anchor_xsi`

The specification of this file is identical to the specification of the `init_parameters` file. The values, however, will be taken as fixed; and they will not be altered during the estimation.

`anchor_tau`

The specification of this file is identical to the specification of the `init_tau` file. The values, however, will be taken as fixed; and they will not be altered during the estimation.

`anchor_reg_coefficients or anchor_beta`

The specification of this file is identical to the specification of the `init_reg_coefficients` file. The values, however, will be taken as fixed; and they will not be altered during the estimation.

`anchor_covariance or anchor_sigma`

The specification of this file is identical to the specification of the `init_covariance` file. The values, however, will be taken as fixed; and they will not be altered during the estimation.

`designmatrix or amatrix`

Specifies an arbitrary item response model. For most ConQuest runs, the model will be specified through the combination of the `score` and `model` statements. However, if more flexibility is required than these statements can offer, then an arbitrary design matrix can be imported and estimated. For full details on the relations between the `model` statement and the design matrix and for rules for defining design matrices, see 'Design Matrices' in Chapter 12 and Volodin and Adams (1995).

`cmatrix`

Specifies an arbitrary model for the estimation of the tau scoring parameters used with the *`scoresfree`* option of the model command. A default scoring design is provided for ConQuest runs using the *`scoresfree`* option, but explicit specification of the Cdesign matrix allows more flexibility. For full details on the relations between the `model` statement and the Cdesign matrix and for rules for defining Cdesign matrices, see ConQuest note "Score Estimation and Generalised Partial Credit Models (revised)".

Options

This command does not have options.

Redirection

`<<` *`filename`*

An import file name must be specified.

Examples

```
import init_parameters << initp.dat;
```

Initial values for item response model parameters are to be read from the file `initp.dat`.

```
import init_parameters << initp.dat;
import anchor_parameters << anch.dat;
```

Initial values for some item response parameters are to be read from the file `initp.dat`, and anchor values for other item response parameters are to be read from `anch.dat`.

```
import designmatrix << design.mat, anchor_c << cov.anc;
```

Imports a design matrix from the file `design.mat` and imports anchor values for the population model covariances from `cov.anc`. Specifying multiple arguments is permissible and is equivalent to using multiple import statements. The arguments and their related redirections are delimited by commas.

GUI Access

File ➔ Import.

Import of each of the file types is accessible as a file menu item.

Notes

(1) After being specified, all file imports remain until a `reset` statement is issued.

(2) If any parameter occurs in both an anchor file and an initial value file, then the anchor value will take precedence.

(3) If any parameter occurs more than once in an initial value file (or files), then the most recent initial value is used.

(4) If any parameter occurs more than once in an anchor file (or files), then the most recent anchor value is used.

(5) Initial value files and anchor values files can contain any subset of the full parameter set.

(6) Importing and exporting cannot occur until the *`estimate`* statement is executed. If a model has been estimated then an `export` statement writes the current estimates to a file. If a model has not been estimated then an export of results will occur immediately after estimation. Also see note 8.

(7) Importing does not result in a change to the internally held estimates until a subsequent estimation command is issued.

(8) You can use the same file names for the import and export files in an analysis: initial values will be read from the files by the `import` statement, and then the `export` statement will overwrite the values in those files with the current parameter estimates as the estimation proceeds or at the end of the estimation.

(9) The number of rows in the imported design matrix must correspond to the number of rows that ConQuest is expecting. ConQuest determines this using a combination of the `model` statement and an examination of the data. The `model` statement indicates which combinations of facets will be used to define generalised items. ConQuest then examines the data to find all of the different combinations; and for each combination, it finds the number of categories. The best strategy for manually building a design matrix usually involves running ConQuest, using a `model` statement to generate a design matrix, and then exporting the automatically generated matrix, using the `designmatrix` argument of the `export` statement. The exported matrix can then be edited as needed before importing it with the `designmatrix` argument of the `import` statement.

(10) Comments can be included in any initial value or anchor value files. Comments are useful for documentation purpose, they are included between the comment delimiters “/*” and “*/”

(11) If a parameter is not identified, ConQuest drops this parameter from the parameter list. This has implications for the parameter sequence numbering in anchor and initial value files. The values in these files must correspond to the parameters numbers **after** removal of non-identified parameters from the parameter list.

itanal

Performs a traditional item analysis for all of the generalised items.

Argument

This command does not have an argument.

Options

`format=`*`type`*

`type` can take the value *`long`*, *`summary`* or *`export`*. If the type is *`summary`* then a compact output that includes all information for each item on a single line is provided. If the type is *`export`* then a complete output is provided but with some omitted formatting. Both *`summary`* and *`export`* formats may facilitate reading of the results into other software. The default is *`long`*.

The export format is provided for reading by other computer programs. Therefore it does not include labelling. The format of the file is as follows.

If there are *k* possible responses to an item the file will contain *k*+3 lines for each generalised item.

Line 1 will contain the number of cases who responded to this item in columns 1 through 6 and the item discrimination in columns 7–11. The remaining columns of the line will contain the item name.

Line 2 contains the item thresholds and the weighted mean square statistics.

Line 3 contains the item delta parameter estimates.

Lines 4 to *k*+3 will contain *k* sets of information, one for each possible response. Columns 1–10contain the response label, columns 11–15 contain the score for the response, columns 16–24 show the number of students who gave the response, column 25–35 give this number as a percentage of the total number of respondents to the item, 36–43 gives the point-biserial for the category, columns 44–58 give a t-test for the point-biserial and matching p-value, columns 59–64 give the mean ability for students giving this response (based upon plausible values), and columns 55–73 give the standard deviation of ability for students giving this response (based upon plausible values). If the model is multidimensional additional columns showing mean and standard deviations of abilities for each extra dimension will be shown.

The *`summary`* format provides a line of information for each generalised item. The information given is restricted to the item label, facility, discrimination, fit and item parameter estimates.

`group=`*`variable`*

An explicit variable to be used as a grouping variable. Results will be reported for each value of the group variable. The variable must have been listed in a previous `group` command.

`estimates=`*`type`*

`type` can take the value `latent`, *`wle`*, *`mle`* or *`eap`*. This option controls the estimator used for the mean and standard deviation of the students that respond in each reported category. The default is *`latent`*.

`filetype=`*`type`*

`type` can take the value *`excel`*, *`xls`*, *`xlsx`* or *`text`*. This option sets the format of the results file. The default is *`text`*.

`matrixout=`*`name`*

`name` is a matrix (or set of matrices) that will be created and will hold the results. These results are stored in the temporary workspace. Any existing matrices with matching names will be overwritten without warning. The contents of the matrices is described in the section ***Matrix Objects Created by Analysis Commands***.

Redirection

`>>` *`filename`*

If redirection to a file is specified, the results will be written to that file. If redirection is omitted, the results will be written to the output window or to the console.

Examples

```
itanal;
```

Performs a traditional item analysis for all of the generalised items and displays the results in the output window or on the console.

```
itanal >> itanal.out;
```

Performs a traditional item analysis for all of the generalised items and writes the results to the file `itanal.out`.

```
itanal estimate=wle, format=export >> itanal.out;
```

Performs a traditional item analysis for all of the generalised items and writes the results to the file `itanal.out` in export format. WLE values are used to estimate category means and standard deviations.

GUI Access

Tables ➔ Export Traditional Item Statistics.

Can be used to produce an export format file of traditional statistics.

Tables ➔ Traditional Item Statistics.

Results in a dialog box. This dialog box is used to select the estimate type, the format and set any redirection.

Notes

(1) The analysis is undertaken for the categories as they exist after applying `recode` statements but before any recoding that is implied by the `key` statement.

(2) Traditional methods are not well-suited to multifaceted measurement. If more than 10% of the response data is missing—either at random or by design (as will often be the case in multifaceted designs)—the test reliability and standard error of measurement will not be computed.

(3) Whenever a `key` statement is used, the `itanal` statement will display results for all valid data codes. If the `key` statement is not used, the `itanal` statement will display the results of an analysis done after recoding has been applied.

(4) If the `export` format is used the results must be redirected to a file.

(5) The `caseweight` command does not influence `itanal` results.

keepcases

List of values for explicit variables that if not matched will cause a record to be dropped from the analysis.

Argument

`list of keep codes`

The *`list of keep codes`* is a comma separated list of values that will be treated as keep values for the subsequently listed explicit variable(s).

When checking for keep codes two types of matches are possible. EXACT matches occur when a code in the data is compared to a keep code value using an exact string match. A code will be regarded as a keep value if the code string matches the keep string exactly, including leading and trailing blank characters. The alternative is a TRIM match that first trims leading and trailing spaces from both the keep string and the code string and then compares the results.

The key words `blank` and `dot`, can be used in the keep code list to ensure TRIM matching of a blank character and a period. Values in the list of codes that are placed in double quotes are matched with an EXACT match. Values not in quotes are matched with a TRIM match.

Options

A comma separated list of explicit variables.

Redirection

Redirection is not applicable to this command.

Examples

```
keepcases 7, 8, 9 ! grade;
```

Retains cases where grade is one of 7, 8 or 9.

```
keepcases M ! gender;
```

Sets M as a keep code the explicit variable gender.

GUI Access

Command → Keep Cases.

Displays a dialog box. Select explicit variables from the list (shift-click for multiple selections) and choose the matching keep value codes. The syntax of the keep code list must match that described above for *`list of keep codes`*.

Notes

(1) Keep values can only be specified for explicit variables.

(2) Complete data records that do not match keep values are excluded from all analyses.

(3) If multiple records per case are used in conjunction with a `pid`, then the `keepcases` applies at the record level not the case level.

(4) See the `missing` command which can be used to omit specified levels of explicit variables from an analysis and the `delete` command which can be used to omit specified levels of implicit variables from an analysis.

(5) See also `dropcases`.

(6) When used in conjunction with SPSS input, note that character strings may include trailing or leading spaces and this may have implications for appropriate selection of a match method.

key

Provides an alternative to the `recode` command that may be more convenient when analysing data from a simple multiple choice or perhaps a partial credit test.

Argument

`codelist`

The *`codelist`* is a string that has the same length as the response blocks given in the `format` statement. When a response block is read, the value of the first response in the block will be compared to the first value in the *`codelist`* argument of any `key` statements. Then the value of the second response in the response block will be compared to the second value in the *`codelist`*, and so forth. If a match occurs, then that response will be recoded to the value given in the *`tocode`* option of the corresponding `key` statement, after all the `key` statements have been read.

If leading or trailing blank characters are required, then the argument can be enclosed in double quotation symbols (`"  "`).

When one or more `key` statements are supplied, any response that does not match the corresponding value in one of the *`codelists`* will be recoded to the value of `key_default`, which is normally *0*. The value of `key_default` can be changed with the `set` command.

If the argument is omitted, then all existing key definitions are cleared.

Options

`tocode`

The value to which matches between the response block and the *`codelist`* are recoded. The column width of the *`tocode`* must be equal to the width of each response as specified in the `format` statement. The *`tocode`* cannot contain trailing blank characters, although embedded or leading blanks are permitted. If a leading blank is required, then the *`tocode`* must be enclosed within double quotation symbols (`"  "`).

Redirection

Redirection is not applicable to this command.

Examples

```
format responses 1-14;
key abcdeaabccabde ! 1;
```

The `format` statement indicates that there are 14 items, with each response taking one column. Any time the first response is coded a, it will be recoded to 1; any time the second response is coded b, it will be recoded to 1; and so on.

```
format responses 1-14 ! rater(2), items(7);
key abcdeaabccabde ! 1;
```

The `format` statement indicates that there are seven items and two raters, with each response taking one column. The recoding will be applied exactly as it is in the first example. Note that this means a different set of recodes will be applied for the items for each rater.

```
format responses 1-14 (a2);
key " a b c d e a a" ! " 1";
```

The `format` statement indicates that there are seven items, with each response taking two columns. Any time the first response is coded a with a leading blank, it will be recoded to 1 with a leading blank. Any time the second response is coded b with a leading blank, it will be recoded to 1 with a leading blank, and so on.

```
format responses 1-14;
key abcdeaabccabde ! 1;
key caacacdeeabccd ! 2;
```

The `format` statement indicates that there are 14 items, with each response taking one column. Any time the first response is coded a, it will be recoded to 1; if it is coded c, it will be recoded to 2. Any time the second response is coded b, it will be recoded to 1; if it is coded a, it will be recoded to 2; and so on.

```
format responses 1-14;
key abcd1111111111 ! 1;
key XXXX2222222222 ! 2;
```

The `format` statement indicates that there are 14 items, with each response taking one column. The item set is actually a combination of four multiple choice and ten partial credit items, and we want to recode the correct answers to the multiple choice items to 1 and the incorrect answers to 0, but for the partial credit items we wish to keep the codes 1 as 1 and 2 as 2. The Xs are inserted in the *`codelist`* argument of the second `key` statement because the response data in this file has no Xs in it, so none of the four multiple choice items will be recoded to 2. While the second `key` statement doesn't actually do any recoding, it prevents the 2 codes in the partial credit items from being recoded to 0, as would have occurred if only one `key` statement had been given.

GUI Access

Command ➔ Scoring➔ Key.

Selecting the key menu item displays a dialog box. This dialog box can be used to build a key command. The syntax requirements for the string to be entered as the Key String are as described above for the *`codelist`*.

Notes

(1) The recoding that is generated by the `key` statement is applied after any recodes specified in a `recode` statement.

(2) Incorrect responses are not recoded to the `key_default` value (0 unless changed by the `set` command) until all `key` statements have been read and all correct-response recoding has been done.

(3) The `key_default` value can only be one character in width. If the responses have a width that is greater than one column, then ConQuest will pad the `key_default` value with leading spaces to give the correct width.

(4) Whenever a `key` statement is used, the `itanal` command will display results for all valid data codes. If the `key` statement is not used, the `itanal` command will display the results of an analysis done after recoding has been applied.

(5) Any missing-response values (as defined by the `set` command argument `missing`) in *`codelist`* will be ignored. In other words, `missing` overrides the `key` statement.

(6) *`tocode`* can be a missing-response value (as defined by the `set` command argument `missing`). This will result in any matches between the responses and *`codelist`* being treated as missing-response data.

kidmap

Produces kidmaps.

Argument

This command does not have an argument.

Options

`cases=`*`caselist`*

A list of case numbers to display. The default is *`all`*.

`estimates=`*`type`*

`type` can take the value *`latent`*, *`wle`*, *`mle`* or *`eap`*. This option controls the estimator that is used for the case location indicator on the map. The default is *`wle`*.

`pagelength=`*`n`*

Sets the length, in lines, of the kidmaps for each case. The default is `60`.

Redirection

```
>> filename
```

If redirection to a file is specified, the results will be written to that file. If redirection is omitted, the results will be written to the output window or to the console.

Examples

```
kidmap;
```

Displays kidmaps for every case in the output window or on the console.

```
kidmap >> kidmap.out;
```

Writes kidmaps for every case to the file `kidmap.out`.

```
kidmap ! cases 1-50, estimate=eap, pagelength=80
   >> kidmap.out;
```

Writes kidmaps for cases 1 to 50 to `kidmap.out`. EAP values are used for case locations and the page length for each map is set to 80 lines.

GUI Access

Tables ➔ Kidmap.

Notes

(1) Case fit statistics are only reported if they are available (see `estimate`)

(2) If the model is multidimensional, a map is prepared for each dimension. Within-item multidimensional items are omitted from the displays

labels

Specifies labels for any or all of the implicit, variables, explicit variables, dimensions and parameters.

Argument

The `labels` statement has two alternative syntaxes. One reads the labels from a file; and one directly specifies the labels.

If the `labels` statement is provided without an argument, then ConQuest assumes that the labels are to be read from a file and that redirection is be provided.

If an argument is provided, it must contain two elements separated by one or more spaces. The first element is the level of the variable (e.g., 1 for item 1), and the second element is the label that is to be attached to that level. If the label contains blank characters, then it must be enclosed in double quotation marks (`"  "`).

Options

The option is only used when the labels are being specified directly.

`variable name`

The variable name to which the label applies. The variable name can be one of the implicit variables or one of the explicit variables or it can be one of `dimensions`, `parameters` or `fitstatistics`. `dimensions` is used to enter labels for the dimensions in a multidimensional analysis, `parameters` is used to enter labels for the parameters in an imported design matrix. `fitstatistics` is used to enter labels for the tests in an imported fit matrix.

Redirection

`<<` *`filename`*

Specifies the name of a file that contains labels. Redirection is not used when you are directly specifying labels.

The label file must begin with the special symbol ===> (a string of three equal signs and a greater than sign) followed by a variable name. The following lines must each contain two elements separated by one or more spaces. The first element is the level, and the second element is the label for that level. If a label includes blanks, then that label must be enclosed in double quotation marks (`"  "`). The following is an example:

```
===> item
1   BSMMA01
2   BSMMA02
3   BSMMA03
4   BSMMA04
5   BSMMA05
```

```
===> rater
1 Frank
2 Nikolai
3 "Ann Marie"
4 Wendy
```

Examples

```
labels << example1.nam;
```

A set of labels is contained in the file example1.nam.

```
labels 1 "This is item one" ! item
```

This gives the label This is item one to level 1 for the variable item.

GUI Access

Command ➔ Labels

Direct label specification is only available using the command line interface.

Notes

(1) The `reset` statement removes all label definitions.

(2) Assigning a label to a level for a variable that already has a label assigned will cause the original label to be replaced with the new label.

(3) There is no limit on the length of labels, but most ConQuest displays are limited in the amount of the label that can be reported. For example, the tables of parameter estimates produced by the `show` statement will display only the first 11 characters of a label.

(4) Labels are not required by ConQuest, but they are of great assistance in improving the readability of any ConQuest printout, so their use is strongly recommended.

let

Creates a ConQuest data object. The data object can be an integer, a real number, a matrix or a string. If the object is a string it is referred to as a token, otherwise it is referred to as a variable.

Argument

`t=`*`string`*

Sets the value of the token named 't' to the string.

or

`t=`*`matrix(value)`*

Declares a matrix variable named 't'. Value must be two integers separated by the character ':'. The two integers are the number of rows and columns respectively.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
let x=10;
```

Sets the token x to the value 10.

```
let path=\w:cycle2\data\;
```

Sets the token path to the value `\w:cycle2\data\`

```
let x=10;
let path=\w:cycle2\data\;
datafile %path%run1.dat;
format responses 1-%x%;
model item;
estimate;
show >> %path%run1.shw;
```

Sets the token x to the value 10 and the token path to the value `\w:cycle2\data\`. In the subsequent code, the tokens contained between the '%' characters are replaced with the corresponding strings.

```
let x=matrix(10:3);
```

Creates variable x, which is a 10 by 3 matrix. It is initialised with zeros.

GUI access

Access to this command through the GUI is not available.

Notes

(1) If a token is defined more than once then the last definition takes precedence.

(2) A `reset all` command clears all tokens.

(3) Tokens implement a simple string substitution; as such they cannot be used until after the let command is executed.

(4) If a batch of submitted code includes both let commands and `dofor` commands, then the `dofor` commands are executed prior to the `let` commands. If large loops (eg greater than 100) contain tokens command parsing may be slow. The `execute` command can be used to force execution of the `let` commands prior to loop execution. This will accelerate command parsing.

(5) The character ';' can be used in the `let` statement through use of the character '|' as a replacement for the ';'. Note that this | character is unavailable in tokens.

(6) The `print` command can be used to display all currently defined variables and tokens.

(7) Tokens can be used in any context. Variables however can only be used in a `compute`, `print` or `scatter` command and as matrix input or matrix output for commands that accept such input and output.

matrixsampler

Draws a sample of matrices that has a set of marginal frequencies (sufficient statistics) that are fixed and defined by the current data set. The `matrixsampler` implements a Markov Chain Monte Carlo algorithm.

Argument

This command does not have an argument.

Options

`sets=`*`n`*

`n` is the number of matrices to sample. The default is *`1000`*.

`burn=`*`n`*

`n` is the number of matrices to sample and then discard before the first retained matrix. The default is *`1000`*.

`step=`*`n`*

`n` is the number of matrices to sample and then discard before each retained matrix. The default is *`64`*.

`filetype=`*`type`*

`type` can take the value *`spss`*, *`excel`*, *`xls`*, *`xlsx`* or *`text`*. This option sets the format of the results file. The default is *`text`*.

`manyout=`*`reply`*

`reply` can be *`yes`* or *`no`*. If *`yes`*, an output file is created for each sampled matrix. If `manyout=`*`no`*, a single file containing all matrices is produced. The default value is *`no`*.

`results=`*`name`*

`name` is a matrix that will be created and will hold selected summary statistics for the sampled matrices. The content of the matrices is described in the section ***Matrix Objects Created by Analysis Commands***.

Redirection

`>> filename`

If redirection to a file is specified, the results will be written to that file in the format specified by the `filetype` option. If `manyout` is specified then multiple files using the name provided with a file number addition will be produced. If redirection is omitted, then no results will be written.

Examples

```
matrixsampler ! filetype=spss >> sampler.sav;
```

Samples 1000 matrices and writes the results to the SPSS system file `sampler.sav`.

```
matrixsampler ! filetype=spss, manyout=yes,
   results=correlations >> sampler.sav;
```

Samples 1000 matrices and writes them to 1000 separate SPSSsystem files (`sampler_1.sav` to `sampler_1000.sav`). A matrix variable with the name correlations is added to the workspace.

GUI access

Access to this command through the GUI is not available.

Notes

(1) The matrixsampler can take a considerable amount of time, especially with large numbers of items and/or cases.

mh

Reports Mantel-Haenszel statistics.

Argument

This command does not have an argument.

Options

`gins=`*`ginlist`*

`ginlist` is a list of generalised item numbers. The default is *`all`*.

`bins=`*`n`*

`n` is the number of groups cases that are used for the raw data.

`estimates=`*`type`*

`type` is one of *`wle`*, *`mle`*, *`eap`* and *`latent`*. This option sets the type of case estimate that is used for constructing the raw data. The default is *`latent`*.

`group=`*`variable`*

An explicit variable to be used as grouping variable. Raw data plots will be reported for each value of the group variable. The variable must have been listed in a previous `group` command.

`reference=`*`variable`*

The specification of the reference group used to report Mantel-Haenszel. The variable must have been the value from the group variable.

`mincut=`*`k`*

`k` is the logit cut between the first and second groups of cases. The default is *−5*.

`maxcut=`*`k`*

`k` is the logit cut between the last and second last groups of cases. The default is *5*.

`bintype=`*`size/width`*

Specifies that the bins are either of equal *`size`* (in terms of number of cases) or of equal *`width`* (in terms of logits). The default is *`size`*. If `bintype=`*`size`*, then the `mincut` and `maxcut` options are ignored.

`keep=`*`keeplist`*

`keeplist` is a list of group identification labels separated by colons. Only those values in the *`keeplist`* will be retained.

`drop=`*`droplist`*

`droplist` is a list of group identification labels separated by colons. Those values in the *`droplist`* will be omitted.

Redirection

Redirection is not applicable to this command.

Examples

```
mh ! group=gender, reference=M;
```

Performance Mantel-Haenszel analysis based upon gender with '`M`" as the reference category.

```
mh ! group=gender, reference=M, bins=5, estimates=wle;
```

Performance Mantel-Haenszel analysis based upon gender with '`M`" as the reference category, and using five groups based upon case WLE estimates.

GUI access

Analysis ➔ Mantel-Haenszel

Results in a dialog box. You can select to report Mantel-Haenszel statistics for all or a subset of items. The command options can then be entered following the syntax guidelines given above.

Notes

(1) The Mantel-Haenszel statistic can also be accessed in conjunction with the `plot` command.

missing

Sets missing values for each of the explicit variables.

Argument

A list of comma separated values that will be treated as missing values for the subsequently listed explicit variable(s).

When checking for missing codes two types of matches are possible. EXACT matches occur when a code in the data is compared to a missing code value using an exact string match. A code will be regarded as missing if the code string matches the missing string exactly, including leading and trailing blank characters. The alternative is a TRIM match that first trims leading and trailing spaces from both the missing string and the code string and then compares the results.

The key words, `blank` and `dot`, can be used in the missing code list to ensure TRIM matching of a blank character and a period. Values in the list of codes that are placed in double quotes are matched with an EXACT match. Values not in quotes are matched with a TRIM match.

Option

A comma separated list of explicit variables.

Redirection

Redirection is not applicable to this command.

Examples

```
missing blank, dot, 99 ! age;
```

Sets blank, dot and 99 (all using a trim match) as missing data for the explicit variable `age`.

```
missing blank, dot, " 99" ! age;
```

Sets blank, and dot (using a trim match) and 99 with leading spaces (using an exact match) as missing data for the explicit variable `age`.

GUI Access

Command ➔ Missing Values.

Select explicit variables from the list (shift-click for multiple selections) and choose the matching missing value codes. The syntax of the missing code list must match that described above for *`list of missing codes`*.

Notes

(1) This command control setting missing values for explicit variables only. For setting the missing values for response data see the `respmiss` option of the `set` command and the `recode` command.

model

Specifies the item response model that is to be used in the estimation. A `model` statement must be provided before any estimation can be undertaken.

Argument

The `model` statement argument is a list of additive terms containing implicit and explicit variables. It provides an expression of the effects that describe the difficulty of each of the responses. The argument `rater+item+item*step`, for example, consists of three terms: `rater`, `item` and `item*step`. The `rater` and `item` terms indicate that we are modelling the response probabilities with a main effect for the rater (their harshness, perhaps) and a main effect for the item (its difficulty). The third term, an interaction between `item` and `step`, assumes that the items we are modelling are polytomous and that the step transition probabilities vary with `item` (See note (1)).

Terms can be separated by either a plus sign (+) or a minus sign (-) (a hyphen or the minus sign on the numeric keypad), and interactions between more than two variables are permitted.

Options

`rasch, pairwise, scoresfree or bock`

The only permissible options are `rasch`, `pairwise, scoresfree and bock`. `rasch` yields a model where scores are fixed. `pairwise` results in the use of a BLT pairwise comparison mode. `scoresfree` results in a generalised model in which item scores are estimated for each items and `bock` results in a generalised model in which scores are estimated for each response category. The default is `rasch`.

Redirection

Redirection is not applicable to this command.

Examples

```
model item;
```

The `model` statement here contains only the term `item` because we are dealing with single-faceted dichotomous data. This is the simple logistic model.

```
model item + item * step;
```

This is the form of the `model` statement used to specify the partial credit model. In the previous example, all of the items were dichotomous, so a model statement without the `item*step` term was used. Here we are specifying the partial credit model because we want to analyse polytomous items or perhaps a mixture of dichotomous and polytomous items.

```
model item + step;
```

In this example, we assume that `step` doesn't interact with `item`. That is, the step parameters are the same for all items. Thus we have the rating scale model.

```
model rater + item + rater * item * step;
```

Here we are estimating a simple multifaceted model. We estimate `rater` and `item` main effect and then estimate separate step-parameters for each combination of `rater` and `item`.

```
model item - gender + item * gender;
```

The `model` statement that we are using has three terms (`item`, `gender`, and `item*gender`). These three terms involve two facets, `item` and `gender`. As ConQuest passes over the data, it will identify all possible combinations of the `item` and `gender` variables and construct *generalised* items for each unique combination. The `model` statement asks ConQuest to describe the probability of correct responses to these generalised items using an item main effect, a gender main effect and an interaction between item and gender.

The first term will yield a set of item difficulty estimates, the second term will give the mean abilities of the male and female students respectively, and the third term will give an estimate of the difference in the difficulty of the items for the two gender groups. This term can be used in examining DIF. Note that we have used a minus sign in front of the `gender` term. This ensures that the gender parameters will have the more natural orientation of a higher number corresponding to a higher mean ability (See note (2)).

```
model rater + criteria + step;
```

This `model` statement contains three terms (`rater`, `criteria` and `step`) and includes main effects only. An interaction term `rater*criteria` could be added to model variation in the difficulty of the criteria across the raters. Similarly, we have applied a single step-structure for all rater and criteria combinations. Step structures that were common across the criteria but varied with raters could be modelled by using the term `rater*step`, step structures that were common across the raters but varied with criteria could be modelled by using the term `criteria*step`, and step structures that varied with rater and criteria combinations could be modelled by using the term `rater*criteria*step`.

```
model essay1 – essay2 ! pairwise;
```

Results in a pairwise comparison model where it is assumed that the explicit variables `essay1` and `essay2` provide information on what has been compared.

```
score (0,1,2,3) (0,1,2,3) ( ) ( ) ( ) ( ) ! item (1-6);
score (0,1,2,3) ( ) (0,1,2,3) ( ) ( ) ( ) ! item (7-13);
score (0,1,2,3) ( ) ( ) (0,1,2,3) ( ) ( ) ! item (14-17);
score (0,1,2,3) ( ) ( ) ( ) (0,1,2,3) ( ) ! item (18-25);
score (0,1,2,3) ( ) ( ) ( ) ( ) (0,1,2,3) ! item (26-28);
model item + item * step;
```

The `score` statement indicates the number of dimensions in the model. The model that we are fitting here is a partial credit model with five dimensions, as indicated by the five *score* lists in the `score` statements. For further information, see the `score` command.

GUI Access

Command ➔ Model.

This dialog box can be used to build a model command. Select an item from the list and add it to the model statement.

Notes

(1) The `model` statement specifies the formula of the log odds ratio of consecutive categories for an item. For example, we supply the model statement

```
model rater + item + rater * item * step;
```

If we then use P_{nrik} to denote the probability of the response of person *n* to item *i* being rated by rater *r* as belonging in category *k*, then the model above corresponds to

$$\log (P_{nrik}/P_{nrik-1}) = \theta_n - (\rho_r + \delta_i + \tau_{irk}),$$

where θ_n is person ability; ρ_r is rater harshness; δ_i is item difficulty; and τ_{irk} is the step parameter for item *i,* rater *r*, and category *k*. Similarly, if we use the `model` statement

```
model rater + item + rater*item*step;
```

then the corresponding model will be

$$\log (P_{nrik}/P_{nrik-1}) = \theta_n - (-\rho_r + \delta_i + \tau_{irk}).$$

(2) The signs indicate the orientation of the parameters. A plus sign indicates that the term is modelled with difficulty parameters, whereas a minus sign indicates that the term is modelled with easiness parameters.

(3) In 'The Structure of ConQuest Design Matrices' in Chapter 12, we describe how the terms in the `model` statement argument result in different versions of the item response model.

(4) The `model` statement can be used to fit different models to the same data. The fitting of a multidimensional model as an alternative to a unidimensional model can be used as an explicit test of the fit of data to a unidimensional item response model. The deviance statistic can be used to choose between models. Fit statistics can be used to suggest alternative models that might be fit to the data.

(5) When a partial credit model is being fitted, all score categories between the highest and lowest categories must contain data. (This is not the case for the rating scale model.) See Chapter 9 for an example and further information.

(6) If ConQuest is being used to estimate a model that has within-item multidimensionality, then the set command argument constraints=cases must be provided. ConQuest can be used to estimate a within-item multidimensional model without constraints=cases. This will, however, require the user to define and import a design matrix. The comprehensive description of how to construct design matrices for multidimensional models is beyond the scope of this manual.

(7) A `model` statement must be supplied even when a model is being imported. The imported design matrix replaces the ConQuest generated matrix. The number of rows in the imported design matrix must correspond to the number of rows in the ConQuest-generated design matrix. In addition, each row of the imported matrix must refer to the same category and generalised item as those to which the corresponding row of the ConQuest-generated design matrix refers. ConQuest determines this using a combination of the model statement and an examination of the data. The `model` statement indicates which combinations of facets will be used to define generalised items. ConQuest then examines the data to find all of the different combinations; and for each combination, it finds the number of categories.

(8) Pairwise models are restricted in their data layout. The format must include at least two explicit variables in addition to the responses. The two explicit variables given in the model describe the objects that are being compared through the matching set of responses. If the first listed variable in the `model` statement is judged “better” than the second then a response of one is expected, if the second listed variable in the model statement is judged “better” than a response of zero is expected.

(9) If a `scoresfree` model is selected then constraints must be set to cases.

plot

Produces a variety of graphical displays.

Argument

`plot type`

`plot type` takes one of the values in the following list and indicates the type of plot that is to be produced.

`icc`

Item characteristic curves (by score category).

`mcc`

Item characteristic curves (by response category).

`ccc`

Cumulative item characteristic curves.

`conditional`

Conditional item characteristic curves.

`expected`

Item expected score curves.

`tcc`

Test characteristic curve.

`iinfo`

Item information function.

`tinfo`

Test information function.

`wrightmap`

Wright map.

`ppwrightmap`

Predicted probability Wright map.

`infomap`

Test information function plotted against latent distribution.

`loglike`

Log of the likelihood function.

Options

`gins=`*`ginlist`*

`ginlist` is a list of generalised item numbers. For the arguments; `icc, ccc, expected,` and `iinfo` one plot is provided for each listed generalised item. For the arguments `tcc` and `tinfo` a single plot

is provided with the set of listed items treated as a *test*. The default is `all`.

`bins=`*`n`*

`n` is the number of groups of cases that are used for the raw data. The default is *`60`* for the Wright Maps and *`10`* for all other plots. For `loglike` it is the number of points to plot.

`mincut=`*`k`*

For the arguments; `icc, ccc, expected,` and `iinfo` *`k`* is the logit cut between the first and second groups of cases. For the arguments `tcc` and `tinfo` is the minimum value for which the plot is drawn. The default is *`-5`*.

`maxcut=`*`k`*

For the arguments; `icc, ccc, expected,` and `iinfo` *`k`* is the logit cut between the last and second last groups. For the arguments `tcc` and `tinfo` is the maximum value for which the plot is drawn. The default is *`5`*.

`minscale=`*`k`*

Specifies the minimum value (*k)* for which the plot is drawn. If this command is not used, the minimum value will be calculated automatically. In `infomap`, this option specifies the minimum value for the vertical axis of the latent distribution.

`maxscale=`*`k`*

Specifies the maximum value (*k*) for which the plot is drawn. If this command is not used, the maximum value will be calculated automatically. In `infomap`, this option specifies the maximum value for the vertical axis of the latent distribution.

`bintype=`*`size/width`*

Specifies that the bins are either of equal size (in terms of number of cases) or of equal width (in terms of logits). The default is *`size`*. If `bintype=`*`size`*, then the mincut and maxcut options are ignored. `bintype=`*`width`* is not available for Wright Maps.

`raw=`*`reply`*

Controls display of raw data. If *`reply`* is *`no`* the raw data is not shown in the plot. If *`reply`* is *`yes`* the raw data is shown in the plot. The default is *`yes`*.

`table=`*`reply`*

If *`reply`* is *`yes`* a data table accompanying each plot is written to the output window. The data table includes a test of fit of the empirical and modelled data. The default is *`no`*.

`legend=`*`reply`*

If *`reply`* is *`yes`* legend is supplied. The default is *`yes`* for Wright Maps and *`no`* for all other plots.

`overlay=`*`reply`*

For the arguments: `icc`, `mcc`, `ccc`, expected, conditional and `iinfo` if *`reply`* is *`yes`* the set of requested plots are shown in a single window. If *`reply`* is *`no`* the set of requested plots are each shown in a separate window.

For the argument `infomap`, in conjunction with `group`, `keep`, and `drop` options, if *`reply`* is *`yes`* the requested plots for the specified groups are plotted against the information function on the same plot.

For the arguments `tcc` and `tinfo` if *`reply`* is *`yes`* the requested plots are displayed in the current active plot window. If no window is currently active a new one is created. If *`reply`* is *`no`* the requested plot is shown in a new separate window. The default is *`no`*.

This option is not available for Wright Maps.

`estimates=`*`type`*

`type` is one of *`wle`*, *`mle`*, *`eap`* and *`latent`*. This option sets the type of case estimate that is used for constructing the raw data. The default is *`latent`*. This option is ignored for the arguments `tcc`, `iinfo` and `tinfo`.

`group=`*`variable`*

An explicit variable to be used as grouping variable. Raw data plots will be reported for each value of the group variable. The variable must have been listed in a previous `group` command.

`mh=`*`variable`*

The specification of the reference group used to report Mantel-Haenszel. The `variable` must have been listed as a group variable. The `table` option under `plot` command must set to *`yes`* in order to show the Mantel-Haenszel statistics and it can only be used in conjunction with the arguments `ccc`, `mcc`, `ccc`, `conditional` and `expected`. The default is *`no`*.

`keep=`*`keeplist`*

`keeplist` is a list of group identification labels separated by colons. Only those values in the *`keeplist`* will be retained in plots. This option can only be used in conjunction with a `group` option and cannot be used with drop.

`drop=`*`droplist`*

droplist is a list of group identification labels separated by colons. Those values in the droplist will be omitted from plots. This option can only be used in conjunction with a group option and cannot be used with `keep`.

`bydimension=`*`reply`*

Only applicable to Wright Maps. If reply is yes a plot is supplied for each dimension. If reply is no, all dimensions are printed on a single plot.

`ginlabels=`*`reply`*

Only applicable to Wright Maps. If *`reply`* is *`yes`* each generalised item is labelled. If *`reply`* is *`no`* the labels are suppresed. The default is *`yes`*.

`order=`*`reply`*

Only applicable to Wright Maps. If *`reply`* is *`value`* generalised items are ordered by estimate value. If reply is *`entry`* generalised items are ordered by sequence number. The default is *`entry`*.

`series=`*`reply`*

Only applicable to Wright Maps. If *`reply`* is *`all`*, a single series is used for display of item parameter estimates. If *`reply`* is *`gin`*, a series is provided for each generalised item. If *`reply`* is *`gingroup`* a series is provided for each defined gingroup. If *`reply`* is *`level`* a series is provided for each level of response and if *`reply`* is *`dimension`* a series is provided for the generalised items allocated to each dimension. Generalised items are ordered by sequence number. The default is *`all`*.

`filesave=`*`reply`*

Controls saving of plots to a file. If *`reply`* is *`yes`* each plot is saved to a file. The default is *`no`*, unless redirection is used, in which case the default is *`yes`*.

`showplot=`*`reply`*

If *`reply`* is *`no`* the production of plots is suppressed. The default is *`yes`*.

`xsi=`*`n`*

`n` is the item location parameter number for which the likelihood is to be plotted. This option is only applicable for the `loglike` argument.

`tau=`*`n`*

`n` is the scoring parameter number for which the likelihood is to be plotted. This option is only applicable for the `loglike` argument.

`beta=`*`n1:n2`*

`n1` is the dimension number and *`n2`* is the variable number for the regression parameter for which the likelhood is to be plotted. This option is only applicable for the `loglike` argument.

`sigma=`*`n1:n2`*

`n1` and *`n2`* the dimensions references for the (co)variance parameter for which the likelihood is to be plotted. This option is only applicable for the `loglike` argument.

Redirection

```
>> filename
```

The name or pathname (in the format used by the host operating system) is appended to the front of the plot window name and plots a written to a file in PNG graphics file format. If no redirection is provided and `filesave=yes`, plots will be saved to the working directory with the plot window name.

Examples

```
plot icc;
```

Plots item characteristics curves for all generalised items in separate windows.

```
plot icc ! gins=1-4:7;
```

Plots item characteristics curves for generalised items 1, 2, 3, 4 and 7 in separate windows.

```
plot icc ! gins=1-4:7, raw=no, overlay=yes;
```

Overlays item characteristics curve plots for generalised items 1, 2, 3, 4 and 7 in a single window and does not show raw data.

```
plot tcc ! gins=1-4:7, mincut=-10, maxcut=10;
```

Plots a test characteristic curve, assuming a test made up of items 1, 2, 3, 4 and 7 and uses ability range from –10 to 10.

```
plot tcc ! gins=1-6, mincut=-10,maxcut=10;
plot tcc ! gins=7-12, mincut=-10, maxcut=10, overlay=yes;
```

Displays two test characteristic curves in the same plot. One for the first six items and one for items 7 to 12.

```
plot infomap ! minscale=-4, maxscale=4;
plot infomap ! minscale=-4, maxscale=4, overlay=yes,
   group=country, keep="country2";
```

Displays two latent distributions against the test information function on the same plot. The first latent distribution is for all students. The second distribution is for students in country2. The plot uses latent ability range from -4.0 to +4.0 which is the vertical scale for the latent distribution.

GUI Access

The various plot types are accessed through the items in the Plot menu.

Notes

(1) For dichotomous items the first category is not plotted in the item characteristic curve plot.

(2) The last category is not plotted for cumulative item characteristic curves.

(3) The item thresholds and item parameters estimates are displayed for the plotted generalised item.

(4) If a `pairwise` model has been estimated the only plot available is `wrightmap`.

(5) Fit statistics are provided if (a) they have been estimated and (b) if the model is of the form x+x*step.

(6) The horizontal axis in `infomap` does not have the same scale in either side of the vertical axis, which is why it is not labelled. The total area under the latent distribution is 1.0. The horizontal scale for the latent distribution side of the horizontal axis is set so that the bin with the largest frequency just fits. The test information function is then scaled to have the same maximum. The total area under the test information function is equal to the number of score points.

print

Displays the contents of defined variables and tokens.

Argument

`List of variables, tokens,` a *`quoted string,`* or a valid *`compute expression`*

The listed variables, tokens or the quoted string is printed to the screen, or if requested to a file.

If the *`list of variables`* is omitted then the names of all available variables and the amount of memory they are using is listed.

Option

`filetype=`*`type`*

`type` `can take the value` *`spss`*, *`excel`*, *`xls`*, *`xlsx`* or *`text`*. This option sets the format of the results file. The default is for the display to be directed to the screen. If `filetype` is specified, a name for the output file should be given using redirection. If `filetype` is specified and no redirection is provided, it will result in an error message.

`decimals=`*`n`*

`n` is an integer value that sets the number of decimal places to display when printing to text. The decimal option is ignored for outputs to SPSS and Excel.

Redirection

`>>` *`filename`*

If redirection into a file is specified, the results will be written to that file. If redirection is omitted the results will be written to the output window or to the console. If no redirection is provided and `filetype` has been specified, it will result in an error.

Examples

```
print item;
```

Prints the contents of the variable or token, `item`.

```
print "Hello World"
```

Prints the text: `Hello World`.

```
print;
```

Prints the names of all variables and the memory they consume and all tokens.

```
print counter(10)*y;
```

Prints the content of the result of the computation counter(10)*y.

GUI Access

Workspace ➔ Tokens and Variables.

Displays a dialog box with the available tokens and variables. The dialog box can be used to print the values of the selected token/variable. A "Columns label" window displays the names for each column of the printed/saved output.

If the selected variable is a matrix you can save the values on a file. Available formats for saving files are text, Excel (.xls or .xlsx) or SPSS.

Notes

(1) The `filetype` option and redirection are only available for matrix variables.

put

Saves a system file.

Argument

This command does not have an argument.

Options

This command does not have any options.

Redirection

```
>> mysysfile.sys
```

`mysysfile.sys` is the name of a ConQuest system file that will be created. Reading this file with the get command allows the current session to be continued at a later time.

Example

```
put >> mysysfile.sys;
```

Saves the system file `mysysfile.sys`.

GUI Access

File ➔ Save System file.

Notes

No notes.

quit

Terminates the program. `exit` has the same effect.

Argument

This command does not have an argument.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
quit;
```

ConQuest terminates. All ConQuest system values will be set to their default values when you next run the application.

GUI Access

File ➔ Exit.

Notes

(1) If you execute a command file that includes a `quit` statement, the `quit` statement will terminate the ConQuest program. If you do not wish to terminate the ConQuest program at that point, omit the `quit` statement from the command file.

recode

Changes raw response data to a new set of values for implicit variables.

Argument

`(from1 from2 from3…) (to1 to2 to3…)`

The argument consists of two code lists, the *from* codes list and the *to* codes list. When ConQuest finds a response that matches a *from* code, it will change (or recode) it to the corresponding *to* code. The codes in either list can be comma-delimited or space-delimited.

Options

list of variables and their levels

Specifies the items to which the recoding in the *to* codes list should be applied. The default is to apply the recoding to all responses.

Redirection

Redirection is not applicable to this command.

Examples

```
recode (a b c d) (0 1 2 3);
```

Recode a to 0, b to 1, c to 2 and d to 3. The recode is applied to all responses.

```
recode (a,b,c,d) (0,1,2,3) ! item (1-10);
```

Recode a to 0, b to 1, c to 2 and d to 3. The recode is applied to the responses to items 1 through 10.

```
recode (" d" " e") (3 4);
```

Recode d with a leading blank to 3, and recode e with a leading blank to 4. If you want to use leading, trailing or embedded blanks in either code list, they must be enclosed in double quotation marks (" ").

```
recode (1 2 3) (0 0 1) ! rater (2, 3, 5-8);
```

The above example states that for raters 2, 3, 5, 6, 7, and 8, recode response data 1 to 0, 2 to 0, and 3 to 1.

```
recode (e,f) (d,d) ! essay (A,B), school("  1001",
    "  1002", "  1003");
```

Recode responses e and f to d when the essays are A and B *and* the school code is 1001, 1002 or 1003 preceded by two blanks. The options here indicate an AND criteria.

```
recode (e,f) (d,d) ! essay (A,B);
recode (e,f) (d,d) ! school("  1001","  1002",
   "  1003");
```

Recode responses `e` and `f` to `d` when the essays are `A` or `B` *or* when the school code is `1001`, `1002` or `1003` preceded by two blanks *or* when both criteria apply. The use of the two `recode` statements allows the specification of an OR criteria.

GUI Access

Command ➔ Recode.

The list will show all currently defined implicit variables. To recode for specific variables select them from the list (shift-click for multiple selections) and select Specify Recodes. A recode dialog box will then be displayed. A *from* codes list *to* codes list can then be entered following the syntax guidelines given above.

Notes

(1) The length of the *`to`* codes list must match the length of the *`from`* codes list.

(2) `recode` statement definitions stay in effect until a `reset` statement is issued.

(3) If a `key` statement is used in conjunction with a `recode` statement, then any `key` statement recoding is applied *after* the `recode` statement recoding. The `recode` statement is only applied to the raw response data as it appears in the response block of the data file.

(4) Any missing-response value (as defined by the `set` command argument `missing`) in the *`from`* code list will be ignored.

(5) Missing-response values (as defined by the `set` command argument `missing`) can be used in the *`to`* code list. This will result in any matches being recoded to missing-response data.

(6) Any codes in the response block of the data file that do not match a code in the *`from`* list will be left untouched.

(7) When ConQuest models the data, the number of response categories that will be assumed for each item will be determined from the number of distinct codes after recoding. If item 1 has three distinct codes, then three categories will be modelled for item 1; if item 2 has four distinct codes, then four categories will be modelled for item 2.

(8) When a partial credit model is being fitted, all score categories between the highest and lowest categories must contain data. (This is not the case for the rating scale model.) The `recode` statement is used to do this. See Chapter 9 for an example and further information.

(9) A `score` statement is used to assign scores to response codes. If no `score` statement is provided, ConQuest will attempt to convert the response codes to scores. If this cannot be done, an error will be reported.

regression

Specifies the independent variables that are to be used in the population model.

Argument

A list of explicit variables to be used as predictors of the latent variable. The list can be comma-delimited or space-delimited. A range of variables can be indicated using the reserved word `to`. The variables can be restricted to particular latent dimensions by replacing dimension numbers in parenthesis after the variable name.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
regression age grade gender;
```

Specifies `age`, `grade` and `gender` as the independent variables in the population model; that is, we are instructing ConQuest to regress latent ability on age, grade and gender.

```
regression ses, y1, y2;
```

Specifies `ses`, `y1` and `y2` as the independent variables in the population model.

```
regression ses to y2;
```

Specifies all `variables` from `ses` to `y2` as independent variables. The variables included from `ses` to `y2` depend on the order given by the user in a previous `format` command (ie if `y1` is listed after `y2` in the `format` command it will not be included in this specification).

```
regression age(2);
```

Regresses dimension two on `age`, but does not regress any other dimensions on age.

```
regression;
```

Specifies a population model that includes a mean only.

GUI Access

Command ➔ Regression Model.

Select regression model variables from the currently defined list of explicit variables (shift-click to make multiple selections).

Notes

(1) Each of the independent variables that are specified in a `regression` statement must take only one value for each measured object (typically a person), as these are 'attribute' variables for each person. For example, it would be fine to use `age` as a regression variable, but it would not make sense to use `item` as a regression variable.

(2) If no `regression` statement is supplied or if no variable is supplied in the `regression` statement, a constant is assumed, and the regression coefficient that is estimated is the population mean.

(3) A `constant` term is always added to the supplied list of regression variables.

(4) If you want to regress the latent variable onto a categorical variable, then the categorical variable must first be appropriately recoded. For example, dummy coding or contrast coding can be used. A variable used in regression must be a numerical value, not merely a label. For example, gender would normally be coded as 0 and 1 so that the estimated regression is the estimated difference between the group means. Remember that the specific interpretation of the latent regression parameters depends upon the coding scheme that you have chosen for the categorical variable. See the `categorise` command.

(5) The `regression` statement stays in effect until it is replaced with another `regression` statement or until a `reset` statement is issued. If you have run a model with regression variables and then want to remove the regression variables from the model, the simplest approach is to issue a `regression` statement with no argument.

(6) If any of the independent variables that are specified in a `regression` statement have missing data, the records are deleted listwise. Because of the cumulative effect of listwise deletion, the overall number of records deleted may increase substantially more than the proportion of missing data in each independent variable as more independent variables are added. This has important consequences in terms of parameter bias, especially if the overall missing data rate substantially exceeds the suggested cut-off values in the literature (ranges from 5–20%, see for example Schafer, 1999 and Peng et al., 2006). The point at which the amount of missing data becomes detrimental will depend on a number of factors including the pattern of missingness, and is beyond the scope of this manual. However, it is recommended in these situations that the user *not* use `regression` or alternatively seek other external methods to handle the missing data (e.g, through multiple imputation, FIML, etc).

reset

Resets ConQuest system values to their default values. It should be used when you wish to erase the effects of all previously issued commands.

Argument

Can be the word *`all`* or *`blank`*. When used without *`all`*, tokens and variables are not cleared.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
reset;
```

Reset all values except tokens and variables.

```
reset all;
```

Reset all values including tokens and variables.

GUI Access

Workspace ➔ Reset.
Workspace ➔ Reset All.

Notes

(1) The `reset` statement can be used to separate jobs that are put into a single command file. The `reset` statement returns all values to their defaults. Even though many values may be the same for the analyses in the command file, we advise resetting, as you may be unaware of some values that have been set by the previous statements.

(2) When a `reset` statement is issued, the output buffer is cleared automatically, with no prior warning.

scatter

Produces a scatter plot of two variables.

Argument

`x,y`

x and *y* must be two existing matrix variables, or a valid compute expression. The matrix variables must each have one column and an equal number of rows. In the case where the compute expression is used, the result must have one column and an equal number of rows to the other variable or expression.

Options

`title=text`

Text to be used as a graph title. The default is *scatter*.

`subtitle=text`

Text to be used as a graph subtitle. The default is *x against y*.

`seriesname=text`

Text to be used as a series name. The default is *x against y*.

`xlab=text`

Text to be used as a series name subtitle. The default is the x-variable name.

`ylab=text`

Text to be used as a series name subtitle. The default is the y-variable name.

`xmin=k`

Specifies the minimum value (*k)* for the horizontal axis. If this option is not used, the minimum value will be calculated automatically.

`ymin=k`

Specifies the minimum value (*k)* for the vertical axis. If this option is not used, the minimum value will be calculated automatically.

`xmax=k`

Specifies the maximum value (*k)* for the horizontal axis. If this option is not used, the maximum value will be calculated automatically.

`ymax=k`

Specifies the maximum value (*k)* for the vertical axis. If this option is not used, the maximum value will be calculated automatically.

`legend=`*`reply`*

If *`reply`* is *`yes`* legend is supplied. The default is *`no`*.

`overlay=`*`reply`*

Results in the scatter plot being overlayed on the existing active plot (if there is one). The default is *`no`*.

`join=`*`type`*

If *`type`* is *`yes`* the points in the plot are joined by a line. The default is *`no`*.

Redirection

`>>` *`filename`*

The name or pathname (in the format used by the host operating system) is appended to the front of the plot window name and plots are written to a file in PNG graphics file format. If no redirection is provided and `filesave=yes`, plots will be saved to the working directory with the plot window name.

Example

```
let a=matrix(14:1);
let b=matrix(14:1);
compute a={-18,16,-7,3,8,-4,6,-5,-9,-4,6,5,-12,-15};
compute b={5,4,9,3,7,-6,-5,1,0,-16,2,-13,-17,5};
scatter a,b ! legend=yes, seriesname=A vs B,
   title=Comparison of A and B;
```

Creates two matrices (a and b) of 14 rows and one column each. Displays a scatter plot of a against b, including a legend with the series name and a title.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) If plots are overlayed the options for the last plot are used for labels and axes ranges.

score

Describes the scoring of the response data.

Argument

(code1 code2 code3…) (score1dim1 score2dim1 score3dim1…) (score1dim2 score2dim2 score3dim2…) …

The first set of parentheses contains a set of codes (the *codes* list). The second set of parentheses contains a set of scores on dimension one for each of those codes (a *score* list). The third set contains a set of scores on dimension two (a second *score* list) and so on. The number of separate codes in the *codes* list indicates the number of response categories that will be modelled for each item. The number of *score* lists indicates the number of dimensions in the model. The codes and scores in the lists can be comma-delimited or space-delimited.

Options

list of variables and levels

Specifies the responses to which the scoring should be applied. The default is to apply the scoring to all responses.

Redirection

Redirection is not applicable to this command.

Examples

```
score (1 2 3) (0 1 2);
```

The code `1` is scored as `0`, code `2` as `1`, and code `3` as `2` for all responses.

```
score (1 2 3) (0 0.5 1.0);
```

The code `1` is scored as `0`, code `2` as `0.5`, and code `3` as `1.0` for all responses.

```
score (a b c) (0 0 1);
```

The code `a` is scored as `0`, `b` as `0` and `c` as `1` for all responses. As there are three separate codes in the *codes* list, the model that will be fitted if this `score` statement is used will have three response categories for each item. The actual model will be an ordered partition model because both the `a` and `b` codes have been assigned the same score.

```
score (a b c) (0 1 2) ! items (1-10);
score (a b c) (0 0 1) ! items (11- 20);
```

The code `a` is scored as `0`, `b` as `1`, and `c` as `2` for items 1 through 10, while `a` is scored `0`, `b` is scored `0`, and `c` is scored `1` for items 11 through 20.

```
score ( a , <b,c>, d) (0,1,2) ! items (1-30);
```

The angle brackets in the code list indicate that the codes `b` and `c` are to be combined and treated as one response category, with a score of `1`. Compare this with the next example.

```
score (a, b, c, d) (0, 1, 1, 2) ! items (1-30);
```

In contrast to the previous example, this `score` statement says that `b` and `c` are to be retained as two separate response categories, although both have the same score of `1`.

```
score (a+," a",b+," b",c+," c") (5,4,3,2,1,0) !
   essay(1,2), rater(A102,B223);
```

The option list can contain more than one variable. This example scores the responses in this fashion for essays 1 and 2 and raters A102 and B223. Double quotation marks are required when a code has a leading blank.

```
score (1 2 3) (0 1 2) (0 0 0) (0 0 0) ! items (1-8,12);
score (1 2 3) (0 0 0) (0 1 2) (0 0 0) ! items (9,13-16,
   18);
score (1 2 3) (0 0 0) (0 0 0) (0 1 2) ! items (10,11,17);
```

To fit multidimensional models, multiple *score* lists are provided. Here, the `score` statement has three *score* lists after the *codes* list, so the model that is fitted will be three-dimensional. Items 1 through 8 and item 12 are on dimension one; items 9, 13 through 16 and 18 are on dimension two; and items 10, 11 and 17 are on dimension three. Because each item is assigned to one dimension only (as indicated by the zeros in all but one of the *score* lists for each `score` statement), we call the model that will be fitted when the above `score` statements are used is a between-item multidimensional model.

```
score (1 2 3) (0 1 2) (  ) ! items (1-8,12);
score (1 2 3) (  ) (0 1 2) ! items (9,13-16,18);
score (1 2 3) (0 1 2) (0 1 2) ! items (10,11,17);
```

If nothing is specified in a set of parentheses in the *score* list, ConQuest assumes that all scores on that dimension are zero. This sequence of `score` statements will result in a two-dimensional model. Items 1 through 8 and item 12 are on dimension one; items 9, 13 through 16 and 18 are on dimension two; and items 10, 11 and 17 are on both dimension one and dimension two. We call models of this type within-item multidimensional. See note (4).

GUI Access

Command ➔ Scoring ➔ Non-Key.

To score for specific variables select them from the list (shift-click for multiple selections) and select Specify Scores. A score dialog box will then be displayed. A *from* codes list *to* codes list can then be entered following the syntax guidelines given above. Scoring needs to be specified for each dimension.

Notes

(1) When estimation is requested, ConQuest applies all recodes and then scores the data. This sequence is independent of the order in which the `recode` and `score` statements are entered.

(2) `Score` statements stay in effect until a `reset` statement is issued.

(3) A `score` statement that includes angle brackets results in the automatic generation of a `recode` statement. For example:

```
score ( a , <b,c>, d) (0,1,2);
```

becomes the equivalent of

```
recode (b,c) (b,b);
score (a,b,d) (0,1,2);
```

and stays in effect until a `reset` statement is issued.

(4) A `score` and `model` statement combination can automatically generate within-item multidimensional models only when the `set` command argument `constraints=cases` is specified. To estimate within-item multidimensional models without setting `constraints=cases`, specify the desired `score` and `model` statements, ignore the warnings that are issued and then supply an imported design matrix.

(5) ConQuest makes an important distinction between response categories and response levels (or scores). The number of response categories that will be modelled by ConQuest for an item is determined by the number of unique codes that exist for that item, after performing all recodes. ConQuest requires a score for each response category. This can be provided via the `score` statement. Alternatively, if the `score` statement is omitted, ConQuest will treat the recoded responses as numerical values and use them as scores. If the recoded responses are not numerical values, an error will be reported.

(6) In a unidimensional analysis, a `recode` statement can be used as an alternative to a `score` statement. See note (5).

(7) The `score` statement can be used to indicate that a multidimensional item response model should be fitted to the data. The fitting of a multidimensional model as an alternative to a unidimensional model can be used as an explicit test of the fit of the data to a unidimensional item response model.

(8) If non-integer scoring is used ConQuest can fit two-parameter models and generalised partial credit.

set

Specifies new values for a range of ConQuest system variables or returns all system values definable through the `set` command to their default values.

Argument

A list of comma-separated arguments. Each argument is described below.

`seed=`*`n`*

Sets the seed that is used in drawing random nodes for use in Monte Carlo estimation method and in simulations runs. *`n`* can be any integer value or the word ‘date’. If date is chosen the seed is the time in seconds since January 1 1970. The default seed is *`1`*.

`n_plausible=`*`n`*

Sets the number of vectors of plausible values to be drawn for each case when a plausible value file is requested in estimation. The default is *`5`*.

`fitdraws=`*`n`*

Sets the number of draws from the posterior distributions that are used in estimating fit statistics. The default is *`5`*.

`p_nodes=`*`n`*

Sets the number of nodes that are used in the approximation of the posterior distributions, which are used in the drawing of plausible values and in the calculation of EAP estimates. The default is *`2000`*.

`f_nodes=`*`n`*

Sets the number of nodes that are used in the approximation of the posterior distributions in the calculation of fit statistics. The default is *`2000`*.

`iterlimit=`*`n`*

Sets the maximum number of iterations for which estimation will proceed without improvement in the deviance. The minimum value permitted is 5. The default value is *`20`*.

`innerloops=`*`n`*

Sets the maximum number of Newton steps that will be undertaken for each item response model parameter in the M-Step. The default value is *`10`*.

`constraints=`*`type`*

Sets the way in which identification constraints are applied. *`type`* can take the values *`smart`*, *`items`*, *`cases`* or *`none`*.

If the constraint is set to *`items`*, then identification constraints will be applied that make the mean of the parameter estimates for each term in the `model` statement (excluding those terms that include `step` zero). For example, the model `item+rater` would be identified by making the average item difficulty zero and the average rater harshness zero. This is achieved by setting the difficulty of the last item on each dimension to be equal to the negative sum of the difficulties of the other items on the dimension.

If the constraint is set to *`cases`*, then constraints will be applied through the population model by forcing the means of the latent variables to be set to zero and allowing all item parameters to be free. If regression anchors are supplied, then the regression parameters will be fixed at the values provided and any unanchored regression coefficients will be fixed to zero. The first term in the `model` statement will not have a constraint imposed, but any additional terms will generate sets of parameter estimates that are constrained to have a zero mean.

If the constraint is set to *`smart`*, then `constraints=cases` will be applied if all regression parameters are found to be anchored; otherwise, `constraints=items` will be used.

The default value is *`items`* if no `constraints` argument is provided.

`warnings=`*`reply`*

`reply` can be *`yes`* or *`no`*. If warnings are set to *`no`*, then messages that do not describe fatal or fundamental errors are suppressed. The default value is *`yes`*.

`keeplastests=`*`reply`*

`reply` can be *`yes`* or *`no`*. If iterations terminate at a non-best solution then setting `keeplastests` to *`yes`* will result in current (non-best) parameter estimates being written retained. The default value is *`no`*.

`logestimates=`*`reply`*

`reply` can be *`yes`* or *`no`*. If a log file is requested, setting `logestimates` to *`yes`* will result in parameter estimates being written to the log file after every iteration. The default value is *`yes`*.

`respmiss=`*`reply`*

Controls the values that will be regarded as missing-response data. *`reply`* can be *`none`*, *`blank`*, *`dot`* or *`both`*. If *`none`* is used, no missing-response values are used. If *`blank`* is used, then blank response fields are treated as missing-response data. If *`dot`* is used, then any response field in which the only non-blank character is a single period (.) is treated as missing-response data. If *`both`* is used, then both the blank and the period are treated as missing-response data. The default is *`both`*.

`zero/perfect=`*`r`*

If maximum likelihood estimates of the cases are requested, then this value is used to compute finite latent ability estimates for those cases with zero or perfect scores. The default value is *`0.3`*.

`mle_criteria=`*n*

The convergence criterion that is used in the Newton-Raphson routine that provides maximum likelihood case estimates. The default is *0.005*.

`key_default=`*n*

The value to which any response that does not match its corresponding value in a `key` statement (and is not a missing-response code) will be recoded. The default is *0*.

`buffersize=`*n*

Number of character that can be accumulated in the output window. The default is *32676*.

`directory=`*directory*

Sets the name of the directory that will be assumed as home directory.

`plotwindows=`*n*

Number of plot windows that can be displayed at one time. The default is *100*.

`mle_max=`*n*

The upper limit for an MLE estimate. The default is *15*.

`echo=`*reply*

reply can be *yes* or *no*. *no* will result suppression of screen display of processing and *yes* will turn screen display on. The default value is *yes*.

`nodefilter=`*p*

Is used when `method=`*gauss* is chosen for estimation. The nodes with the smallest weight are omitted from the quadrature approximation in the estimation. The set of nodes with least weight which add to the proportion p of the density are omitted. This option can dramatically increase the speed for multidimensional models. The default is `p=`*0*.

`uniquepid=`*reply*

reply can be *yes* or *no*. Use *yes* for datasets with unique PIDs (i.e., each record corresponds to only one case and only one PID; see `format` command) to drastically reduce the processing time especially for large datasets. The default value is *no*.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Examples

```
set constraints=cases, seed=20;
```

Sets the identification constraints to `cases` and the seed for the Monte Carlo estimation method to `20`.

```
set;
```

Returns all of the `set` arguments to their default values.

GUI Access

Workspace ➔ Set.

Notes

(1) All of the `set` arguments are returned to their default values when a `set` statement without an argument is issued. If a model has been estimated, then issuing this statement will require that the model be re-estimated before `show` or `itanal` statements are issued.

(2) If the `set` statement has an argument, then only those system variables in the argument will be changed.

(3) The `key_default` value can only be one character in width. If the responses have a width that is greater than one column, then ConQuest will pad the `key_default` value with leading spaces to give the correct width.

(4) If `warnings` is set to *no*, then the output buffer will be automatically cleared, without warning, whenever it becomes full. This avoids having to respond to the 'screen buffer is full' messages that will be displayed if you are running an analysis using the GUI interface.

(5) ConQuest uses the Monte Carlo method to estimate the mean and standard deviation of the marginal posterior distributions for each case. The system value `p_nodes` governs the number of random draws in the Monte Carlo approximations of the integrals that must be computed.

(6) `constraints=cases` must be used if you want ConQuest to automatically estimate models that have within-item multidimensionality. If you want ConQuest to estimate within-item multidimensional models without the use of `constraints=cases`, you will have to define and import your own design matrices. The comprehensive description of how to construct design matrices for multidimensional models is beyond the scope of this manual.

show

Produces a sequence of displays to summarise the results of the estimation.

Argument

`parameters`

Requests displays of the parameter estimates in tabular and graphical form. These results can be written to a file or displayed in the output window or on the console. This is the default, if no argument is provided.

`cases`

Requests parameter estimates for the cases. These results must be written to a file using redirection.

`residuals`

Requests residuals for each case/generalised item combination. These results must be written to a file and are only available for weighted likelihood ability estimates.

For pairwise models, the `residuals` statement requests residuals for each fixed pair-outcome combination. The residuals can be interpreted as prediction errors (i.e., the difference between the observed and the predicted outcomes).

`expected`

Requests expected scores for each case/generalised item combination. These results must be written to a file and are only available for weighted likelihood ability estimates.

Options

`estimates=`*`type`*

`type` can be *`eap`*, *`latent`*, *`mle`*, *`wle`* or *`none`*.

When the argument is `parameters` or no argument is provided, this option specifies what to plot for the case distributions. If `estimates=`*`eap`*, the distribution will be constructed from expected a-posteriori values for each case. If `estimates=`*`latent`*, the distribution will be constructed from plausible values so as to represent the latent distribution. If `estimates=`*`mle`* or *`wle`*, the distribution will be constructed from maximum likelihood or weighted likelihood cases estimates. This provides a representation of the latent population distribution. If `estimates=`*`none`*, then the case distributions are omitted from the `show` output. If no `estimates` option is provided and the `estimate` statement includes `fit=`*`yes`* (explicitly or by default), the default is to use plausible values. If the `estimate` statement includes `fit=`*`no`*, the default is to omit the distributions from the `show` output.

When the argument is `cases`, this option gives the type of estimate that will be written to an output file. (See ‘Redirection’ below for the file formats.) `estimates=`*`none`* cannot be used, and there is no default value. Therefore, you must specify *`eap`*, *`latent`*, *`wle`* or *`mle`* when the argument is `cases`.

`tables=`_`value list`_

If `parameters` output is requested, a total of eleven different tables can be produced. If a specific set of tables is required, then the `tables` option can be used to indicate which tables should be provided. *`value list`* consists of one or more of the integers 1 through 11, separated by colons (:) if more than one table is requested.

The contents of the tables are:

1 A summary showing the model estimated, the number of parameters, the name of the data file, the deviance and the reason that iterations terminated.

2 The estimates, errors and fit statistics for each of the parameters in the item response model.

3 Estimates for each of the parameters in the population model and reliability estimates.

4 A map of the latent distribution and the parameter estimates for each term in the item response model.

5 A vertical map of the latent distribution and threshold estimates for each generalised item.

6 A horizontal map of the latent distribution and threshold estimates for each generalised item.

7 A table of threshold estimates for each generalised item.

8 A table of item parameters estimates for each generalised item.

9 A map of the latent distribution and the parameter estimates for each term in the item response model with items broken out by dimension.

10 A table of the asymptotic error variance/covariance matrix for all parameters

11 Score estimates for each category of each generalised item and the scoring parameter estimates

The default tables are as follows, depending on the model (see `model` command) that is estimated – Rasch models: tables=1:2:3:4; 2PL models: tables=1:2:3:4:11; other models with scores: tables=1:2:3:11; pairwise models: tables=1:2. For multidimensional models (see `score` command), table 9 is also produced as default. For partial credit models, it is useful to include table 5 (which is not produced as default) in the requested tables.

`labelled=`*`reply`*

`reply` can be *`yes`* or *`no`*. `labelled=`*`no`* gives a simple form of the output that only includes a list of parameter numbers and their estimates. `labelled=`*`yes`* gives an output that includes parameter

names and levels for each term in the `model` statement. `labelled=`*`yes`* is the default, except when a design matrix is imported, in which case `labelled=`*`yes`* is not available.

`expanded=`*`reply`*

`reply` can be *`yes`* or *`no`*. This option used in conjunction to table 5 to control the display of the item thresholds. `expanded=`*`yes`* separates the thresholds horizontally so that a new column is given for each item. `expanded=`*`no`* is the default.

`itemlabels=`*`reply`*

`reply` can be *`yes`* or *`no`*. This option is used in conjunction to table 5 to control the display of the item thresholds. `itemlabels=`*`yes`* uses item labels for each generalised item. `itemlabels=`*`no`* is the default.

`pfit=`*`reply`*

`reply` can be *`yes`* or *`no`*. This option is used in conjunction to the argument `cases` and the option `estimates=`*`wle`* and adds person fit statistics to the estimates file. `pfit=`*`no`* is the default.

`filetype=`*`type`*

`type` can take the value *`spss`*, *`excel`*, *`xls`*, *`xlsx`* or *`text`*. This option sets the format of the results file. The default is *`text`*. The *`spss`* option is available if the argument is `cases`, `residuals` or `expected`.

`xscale=`*`n`*

Sets the number of cases to be represented by each 'X' in Wright maps. The default value is a value that ensures that the largest bin uses all available bin space. The value is replaced by the default if the display would not otherwise fit in the available space.

`plotmax=`*`n`*

Sets the maximum logit value for the range of Wright maps.

`plotmin=`*`n`*

Sets the minimum logit value for the range of Wright maps.

`plotbins=`*`n`*

Sets the number of bins used for the range of Wright maps. The default value is *`60`*.

`itemwidth=`*`n`*

Sets the width in characters of the region available for item (facet) display in Wright maps. The default value is *`40`*.

Redirection

`>> filename`

Specifies a file into which the show results are written. If redirection is omitted and the argument is `parameters` or no argument is given, the results are written to the output window or the console. If the argument is `cases`, `residuals` or `expected`, then an output file must be given.

When the argument is `cases`, the format of the file of case estimates is as follows. In describing the format of the files we use *nd* to indicate the number of dimensions in the model.

For plausible values (`estimates=latent`):

> If we use *np* to indicate the number of plausible values, then the format of the plausible value file will be as follows.
>
> It will contain *np*+3 lines for each case that provided a valid response to at least one of the items analysed.
>
> Line 1 will contain the case number (the sequence of the case in the data file being analysed) in columns 1 through 5.
>
> Line 2 to line *np*+1 will each contain *nd* plausible values in the format *nd*(t13, *nd*(f6.2)).
>
> Line *np*+2 will contain *nd* EAP estimates in the format *nd*(f10.5, 1x).
>
> Line *np*+3 will contain *nd* posterior variance estimates in the format *nd*(f10.5, 1x).

For expected a-posteriori estimates (`estimates=eap`):

> The file will contain one line for each case. The line will contain the case number (the sequence number of the case in the data file being analysed), the posterior mean, the posterior standard deviation and the reliability for each dimension. The format is (i5, *nd*(3(f10.5, 1x))).

For maximum likelihood estimates and weighted likelihood (`estimates=mle` or `estimates=wle`):

> The file will contain one line for each case that provided a valid response to at least one of the items analysed. The line will contain the case number (the sequence number of the case in the data file being analysed), the raw score and maximum possible score on each dimension, followed by the maximum likelihood estimate and error variance for each dimension. The format is (i5, *nd*(2(f10.5, 1x)), *nd*(2(f10.5, 1x))). If the `pfit` option is set then an additional column is added containing the case fit statistics. The format is then (i5, *nd*(2(f10.5, 1x)), *nd*(2(f10.5, 1x)), f10.5)

Examples

```
show;
```

Produces displays with default settings and writes them to the output window.

```
show ! estimates=latent >> show.out;
```

Produces displays and writes them to the file `show.out`. Representations of the latent distributions are built from plausible values.

```
show parameters ! tables=1:4, estimates=eap;
```

Produces displays 1 and 4, represents the cases with expected a-posteriori estimates, and writes the results to the output window.

```
show cases ! estimates=mle >> example.mle;
```

Produces the file `example.mle` of case estimates, using maximum likelihood estimation.

```
show cases ! estimates=latent >> example.pls;
```

Produces the file `example.pls` of plausible values.

```
show cases ! estimates=wle, pfit=yes >> example.wle;
```

Produces the file `example.wle` of weighted likelihood estimates and person fit statisics.

```
show residuals ! estimates=wle, pfit=yes >> example.res;
```

Produces the file `example.res` of residuals for each case.

GUI Access

The various displays are accessed through the items in the Tables menu.

Notes

(1) The order in which command statements can be entered into ConQuest is not fixed. There are, however, logical constraints on the ordering. For example, `show` statements cannot precede the `estimate` statement, which in turn cannot precede the `model`, `format` or `datafile` statements.

(2) The tables of parameter estimates produced by the `show` command will display only the first 11 characters of the labels.

(3) The method used to construct the ability distribution is determined by the `estimates` option used in the `show` statement. The `latent` distribution is constructed by drawing a set of plausible values for the cases and constructing a histogram from the plausible values. Other options for the distribution are `eap` and `mle`, which result in histograms of expected a-posteriori and maximum likelihood estimates, respectively.

(4) It is possible to recover the ConQuest estimate of the latent ability correlation from the output of a multidimensional analysis by using plausible values. Plausible values can be produced through the use of the `show` command argument `cases` in conjunction with the option `estimates=latent`.

(5) The `show` statement cannot produce individual tables when an imported design matrix is used.

(6) Neither `wle` nor `mle` case estimates can be produced for cases that had no valid responses for any items on one or more dimension. Plausible values are produced for all cases with complete background data.

(7) Table 10 is only available if empirical standard errors have been estimated. Table 10 is not applicable for pairwise models.

(8) Plausible values and EAP estimates contain stochastic elements and may differ marginally from run to run with identical data.

(9) Showing cases is not applicable for pairwise models.

structural

Fits a structural path model using two-stage least squares.

Argument

The structural statement argument is a list of regression models that are separated by the character "/" (slash). Each regression model takes the form `dependent` on `independent_1`, `independent_2`, …, `independent_n`.

Options

`export=`*`reply`*

`reply` can be *`yes`* or *`no`*. This option controls the format of the output. The export format does not use labelling and is supplied so that results can be read into other software easily. `export=`*`no`* is the default.

`filetype=`*`type`*

`type` can take the value *`xls`*, *`xlsx`*, *`excel`* or *`text`* and it sets the format of the results file. The default is *`text`* when used in conjunction with a file redirection. If no file redirection is given the results are written to the output window.

`matrixout=`*`name`*

`name` is a matrix (or set of matrices) that will be created and will hold the results. These results are stored in the temporary workspace. Any existing matrices with matching names will be overwritten without warning. The contents of the matrices is described in the section ***Matrix Objects Created by Analysis Commands***.

Redirection

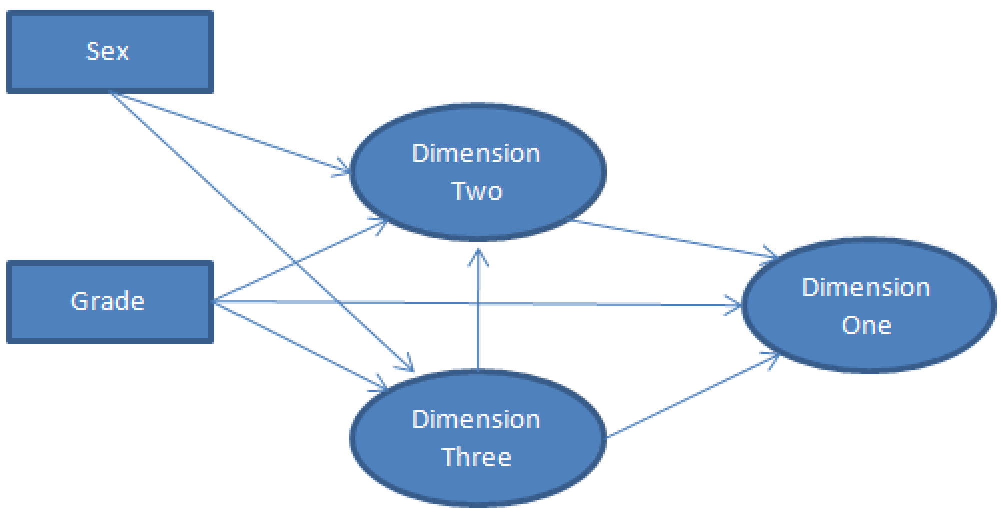
`>>` *`filename`*

Specifies a file into which the `show` results are written.

Example

```
structural /dimension_1 on dimension_2 dimension_3 grade
    /dimension_2 on dimension_3 grade sex
    /dimension_3 on grade sex ! export=yes;
```

Fits the path model shown in the following figure



GUI Access

Analysis ➔ Structural.

Select regression variables from the currently defined list of regression variables and latent variables.

Notes

No notes.

submit

Executes the ConQuest command statements in the file named in its argument.

Argument

`filename`

The name of the text file containing the statements.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
submit example1.cqc
```

Executes the statements in the file `example1.cqc`.

GUI Access

File ➔ Submit Commands.

Notes

(1) `submit` commands can be nested. That means a file of submitted commands can contain a `submit` command.

system

Allows a DOS command to be executed.

Argument

DOS Command

The command to be executed.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
system dir;
```

Shows the contents of the current working directory.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) In the GUI version the results of some operating system commands will be written to a command window that will not stay open after the command has executed.

title

Specifies the title that is to appear at the top of any printed ConQuest output.

Argument

This command does not have an argument.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
title This is a great analysis!;
```

The words `This is a great analysis!` will appear on the top of each ConQuest printout from this analysis.

GUI Access

Command ➔ Title.

Notes

(1) If a title is not provided, the default, *`ConQuest: Generalised Item Response Modelling Software`*, will be used.

while

Allows conditional execution of commands

Argument

```
(logical condition) {
   set of ConQuest commands
   };
```

While logical condition evaluates to `true` the set of ConQuest commands is executed. The commands are not executed if the logical condition does not evaluate to `true`.

The logical condition can be `true`, `false` or of the form *s1* operator *s2*, where *s1* and *s2* are strings and operator is one of the following

```
==   equality
=>   greater than or equal to
>=   greater than or equal to
=<   less than or equal to
<=   less than or equal to
!=   not equal to
>    greater than
<    less than
```

For each of *s1* and *s2* ConQuest first attempts to convert it to a numeric value. The numeric value can be a scalar value, a reference to an existing 1x1 matrix variable or a 1x1 submatrix of an existing matrix variable. A numeric value cannot involve computation.

If *s1* is a numeric value the operator is applied numerically. If not a string comparison occurs between *s1* and *s2*.

Options

This command does not have options.

Redirection

Redirection is not applicable to this command.

Example

```
let x=matrix(20:20);
compute k=1;
compute i=1;
while (i<=20)
{
    for (j in 1:i)
     {
      if (j<i)
       {
        compute x[i,j]=k;
        compute x[j,i]=-k;
        compute k=k+1;
       };

      if (j==i)
       {
        compute x[i,j]=j;
       };
     };
    compute i=i+1;
};
print x;
```

Creates a 20 by 20 matrix of zero values and then fills the lower triangle of the matrix with the numbers 1 to 190, the upper triangle with -1 to -190 and the diagonal with the numbers 1 to 20. The matrix is then printed to the screen.

GUI Access

Access to this command through the GUI is not available.

Notes

(1) There are no limits on the nesting of conditions.

COMPUTE COMMAND OPERATORS AND FUNCTIONS

Operators

The standard binary mathematical operators are: addition (+), subtraction (-), multiplication (*), and division (/). All are available and operate according to their standard matrix definition when applied to conformable matrices. Division by a matrix is treated as multiplication by the matrix inverse. If the operators are applied to non-conformable matrices then the operators return a null matrix excepting when one of the arguments is a double (or 1 by 1 matrix), then the operator is applied element-wise.

The unary negation operator (-) is available and is applied element wise to a matrix.

The exponentiation operator (^) is available but cannot be applied to matrices.

Two special binary mathematical operators are provided for element-wise matrix multiplication (**) and division (//). The ** operator multiplies each of the matching elements of two identically dimensioned matrices. The // operator divides each element of the first matrix by the matching element of the second matrix.

The following 6 logical operators are available: "==", "!=", "<=", ">=", ">", and "<". These operators are applied element-wise to a pair of matrices and return matrices of '1' and '0' with 1 if an element-wise comparison is true and 0 if it is false.

Standard Functions

The following standard functions are available. Each of the functions takes a single matrix argument and is applied element-wise to the matrix.

sqrt	square root of the argument. Argument must be greater ≥ 0
exp	raises e to the power of the argument
log	natural log of the argument
log10	log base 10 of the argument
logit	logit transformation of values between 0 and 1
abs	absolute value of argument
floor	returns largest integer value not greater than the argument
ceil	returns smallest integer value not less than the argument
int	returns integer part of the argument
rnd	rounds the argument to the nearest integer

Accessing Matrix Information

Sub matrices can be extracted from matrices by appending [*rowstart*:*rowend*,*colstart*:*colend*] to the name of a matrix variable, for example m[2:5,5:10]. If all rows are required, *rowstart* and *rowend* can be omitted. If all columns are required, *colstart* and *colend* can be omitted. If a single row is required, *rowend* and the colon ":" can be omitted. If a single column is required, *colend* and the colon ":" can be omitted.

Column and row indexing commence at one. So that, for example, m[10,3] refers to the element in the 10-th row and 3-th column.

Single elements of a matrix can be specified to the left of the equal operator '=' by appending [*row*,*col*] to the name of a matrix variable. Sub matrices cannot be specified to the left of the equal operator '='.

Matrix Manipulation Functions

Two binary operators are available for concatenating matrices. Column concatenation of two matrices, m1 and m2 is performed using m1 |^ m2. In this case, m1 and m2 must have column conformability and the matrix m1 is added under matrix m2. Row concatenation of two matrices, m1 and m2 is performed using m1 -> m2. In this case m1 and m2 must have row conformability and the matrix m1 is added to the left of matrix m2

The following functions are available for manipulating the content of matrices:

counter(*arg*)

Returns a matrix with dimensions *arg* x 1 filled with integers running from 1 to *arg*.

fillmatrix(*arg1*, *arg2*, *arg3*)

Returns a matrix with dimensions *arg1* x *arg2* filled with the value *arg3*.

identity(*arg*)

Returns a matrix of dimension *arg*.

iif(*arg1*, *arg2*, *arg3*)

All three arguments must be matrices of the same dimensions. The result is a matrix where an element takes its value from *arg2* if the matching *arg1* element is '1', otherwise it takes its value from *arg3*.

transpose(*arg*)

Transpose matrix of *arg*.

inv(*arg*)

Inverse of matrix *arg*.

det(*arg*)

Determinant of matrix *arg*.

trace(*arg*)

Trace of matrix *arg*.

rows(*arg*)

Number of rows of matrix *arg*.

cols(*arg*)

Number of columns of matrix *arg*.

min(*arg*)

Number minimum of all elements in matrix *arg*.

max(*arg*)

Number maximum of all elements in matrix *arg*.

sum(*arg*)

Sum of all elements in matrix *arg*.

sum2(*arg*)

Sum of squares of all elements in matrix *arg*.

colcp(*arg*)

Column cross-products, returns a row × row matrix equal to *arg**transpose(*arg*).

rowcp(*arg*)

Row cross-products, returns a column × column matrix equal to transpose(*arg*)**arg*.

rowcov(*arg*)

Row covariance, returns a column × column matrix which is the covariance matrix of the columns.

rowcor(*arg*)

Row correlations, returns a column × column matrix which is the correlation matrix of the columns.

colsum(*arg*)

Returns a row which contains the sum over each of the columns of the *arg.*

rowsum(*arg*)

Returns a vector which contains the sum over each of the rows of the *arg.*

sort(*arg*)

Returns a vector which is contains the rows of *arg* sorted in ascending order. The argument must be a vector.

Random Number Generators

The following random number generators are used. To control the seed use *set* `seed=n`*.*

rnormal(*arg1*, *arg2*)

A random normal deviate with mean *arg1* and standard deviation *arg2*.

rnormalmatrix(arg1, arg2, arg3, arg4)

An *arg3* x *arg4* matrix of random deviates, with mean *arg1* and standard deviation *arg2*.

rlefttnormal(arg)

Deviate from a standard normal left truncated at *arg*.

rrighttnormal(arg)

Deviate from a standard normal right truncated at *arg*.

rchisq(arg)

Chi square deviate with *arg* degrees of freedom.

rinvshisq(arg)

Inverse chi-square deviate with *arg* degrees of freedom.

rbernoulli(arg)

Matrix of Bernoulli variables where *arg* is a matrix of p values.

Matrix Objects Created by Analysis Commands

A number of analysis returns can save their results in a family of matrix objects that are added to the ConQuest variable list. These variables then become available for manipulation or other use.

The commands that can produce matrix variables are: `descriptives`, `estimate`, `fit` and `generate`. For each of these commands the option `matrixout=`*`stem`* is used to request the variables and to set a prefix for their name. The variables produced by each command and their format is provided below.

Descriptives Command

The following four matrices are produced regardless of the estimator option:

descriptives

Number of dimensions by eight, providing for each dimension the dimension number, number of cases, mean, standard deviation, variance, standard error of the mean, standard error of the standard deviation, and standard error of the variance.

percentiles

Number of dimensions by the number of requested percentiles plus two, providing for each dimension the dimension number, number of cases, and then each of the percentiles.

bands

Number of dimensions by twice the number of requested bands plus two, providing for each dimension the dimension number, number of cases, and then proportion in each of the bands follow by standard errors for each of the band proportions.

bench

Number of dimensions by four, providing for each dimension the dimension number, number of cases, proportion below the benchmark and standard error of that proportion.

If `latent` is chosen as the estimator then in addition to the above the following matrices are available:

pv_descriptives

Number of dimensions times number of plausible values by six, providing for each dimension and plausible value, the dimension number, the plausible value number, number of cases, mean, standard deviation, and variance.

pv_percentiles

Number of dimensions times number of plausible values by the number of percentiles plus three, providing for each dimension and plausible value, the dimension number, the plausible value number, number of cases, and then each of the percentiles.

pv_bands

Number of dimensions times number of plausible values by the number of requested bands plus three, providing for each dimension and plausible value, the dimension number, the plausible value number, number of cases, and then proportion in each of the bands.

Estimate Command

The matrices that are produced by estimate depend upon the options chosen. Regardless of the options chosen the following two matrices are produced:

itemparams

A single column of the estimated item location parameters.

history

Number of iterations by total number of estimated parameters plus two. The first column is the iteration number, the second column is the deviance and the remaining columns are for the parameter estimates.

If the `method=jml` option is chosen or `abilities=yes` in conjunction with an MML method then the following two matrices of case estimates are produced.

mle

Number of cases by number of dimensions providing for each case the MLE latent estimate for each case.

wle

Number of cases by number of dimensions providing for each case the WLE latent estimate for each case.

If `abilities=yes` is used in conjunction with an MML method then a matrix of case plausible values is produced.

pvs

Number of cases by number of dimensions times number of plausible values. For the columns the plausible values cycle fastest. For example, if there are three dimensions and two plausible values, column one would contain plausible value one for dimension one, column two would contain plausible value two for dimension one, column three would contain plausible value one for dimension two and so on.

If `ifit=yes` is used (the default) a matrix of item fit values is produced.

itemfit

Number of fit test by four. The four columns are the unweighted T, weighted T, unweighted MNSQ, and weighted MNSQ.

If `pfit=yes` is used a matrix of case fit values is produced.

casefit

Number of cases by one. Providing for each case the unweighted mean square.

If `stderr=empirical` is used (the default for MML) then the estimate error covariance matrix is produced.

estimatecovariances

A number of parameter by number of parameter matrix of estimate error covariances.

If `stderr=quick` is used (the default for JML) then the following estimate error variances matrix are produced.

itemerrors

Number of item parameter estimates by one, providing for each item parameter the estimate variance.

regressionerrors

Number of regression parameters by one, providing for each regression parameter the estimate variance.

covarianceerrors

Number of covariance parameters by one, providing for each regression parameter the estimate variance.

Fit Command

Produces a set of matrices, one for each level of the group used in the `group=option`. The name of the matrix is provided by the `matrixout=option`. Each matrix has dimension number of fit tests by four, providing for each test the unweighted t fit, weighted t fit, unweighted mean square and weighted mean square.

Generate Command

The matrices that are produced by generate depend upon the options chosen. Regardless of the options chosen the following matrix is produced:

items

Number of items by three, providing for each item the item number, category number, and the generated parameter value.

If `scoresdist=` is used then a matrix of scoring parameters is produced.

scores

Number of total item scoring categories by number of dimensions plus two, providing for each item category, the item number, category number and score for each dimension.

If `importnpvs=` is NOT used then the following two matrices are produced:

responses

Number of cases by number of items, providing for each case a response to each item.

cases

Number of cases by number of dimensions plus one, providing for each case a case number and a generated ability for each of the dimensions.

If `importnpvs=` is used then the following matrices of summary statistics are produced for each dimension and group:

statistics

Number of plausible values by three times the number of items plus three. It contains mean raw scores, raw score variances, Cronbach's alpha and then for each item, mean item score and point biserial statistics (biased and unbiased).

Itanal Command

Produces a set of matrices, one for each level of the group used in the `group=option`. The name of the matrix is provided by the `matrixout=option`. The matrices produced are as follows.

counts

Number of items by number of response categories, providing for each item and category number the frequency of responses.

summary

Number of items by two, providing for each item the item-total and item-rest correlations.

stats

Number of items by number of response categories, providing for each item and category the point-biserial correlation with the total score.

means

Number of items by number of response categories by dimension by two, providing for each item, category, and dimension the mean and standard deviation of the first plausible value of the cases who responded to that category.

Matrixsampler Command

Produces a matrix with the name provided in the `results=option`. The matrix produced has a row for each sampled matrix and columns providing the inter-item and item-total correlations.

Structural Command

Produces a set of matrices, one for each regression model and four matrices of sums of squares and cross-products. The name of the matrix is provided by the `matrixout=option`. The matrices produced are as follows.

fullsscp

Square matrix with dimension equal to the total number of variables in the structural model providing the sums of squares and cross-products.

osscp

Square matrix with dimension equal to the number of observed variables (non latent variables) in the structural model providing the sums of squares and cross-products.

losscp

Number of latent by number of observed variables providing the cross-products.

lsscp

Square matrix with dimension equal to the total number of latent variables in the structural model providing the sums of squares and cross-products.

results_eqn*n*

For each regression equation that makes up the structural model the results of the estimation. The cell 1,1 contains the R-squared, and there are additional rows for each independent variable, column one of each of those additional rows is the estimated regression parameter and the second column is its standard error estimate.

LIST OF ILLEGAL CHARACTERS AND WORDS FOR VARIABLE NAMES

Illegal characters to use in tokens names	|	/	$	~

Illegal words to use in matrices, and implicit and explicit variables names					
Words	**Functions**		**Command names**		
on	abs	rchisq	caseweight	filter	model
to	ceil	rinvshisq	categorise	fit	plot
dimensions	colcp	rlefttnormal	clear	for	print
parameters	cols	rnd	codes	format	put
fitstatistics	colsum	rnormal	compute	function	quit
step	counter	rnormalmatrix	datafile	generate	recodes
steps	det	rowcor	delete	get	regression
category	exp	rowcov	descriptives	gingroup	reset
variables	fillmatrix	rowcp	directory	group	scatter
tokens	floor	rows	display	if	scores
all	identity	rowsum	doif	import	set
	int	rrighttnormal	dropcases	itanal	show
	inv	sort	else	keepcases	structural
	log	sqrt	endif	key	submit
	log10	sum	equivalence	kidmap	system
	logit	sum2	estimates	labels	systemclean
	max	trace	execute	let	timerstart
	min	transpose	exit	matrixsampler	timerstop
	rbernoulli	rchisq	export	mh	title
			facets	missing	while