LG-Syntax™ User’s Guide:

Manual for Latent GOLD® 5.0 Syntax Module

October 31, 2013

Jeroen K. Vermunt

Jay Magidson

---

For more information about Statistical Innovations Inc. please visit our website at http://www.statisticalinnovations.com or contact us at

Statistical Innovations Inc.
375 Concord Avenue, Suite 007
Belmont, MA 02478
e-mail: will@statisticalinnovations.com

Latent GOLD, LG-Syntax, and LG-Equations are trademarks of Statistical Innovations Inc.
Windows is a trademark of Microsoft Corporation.
SPSS is a trademark of SPSS, Inc.
Other product names mentioned herein are used for identification purposes only and may be trademarks of their respective companies.

All rights reserved.

No part of this publication may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission from Statistical Innovations Inc.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>The LG-Syntax User Interface</td>
</tr>
<tr>
<td>2.1</td>
<td>Setting up a Syntax session and saving Syntax models</td>
</tr>
<tr>
<td>2.2</td>
<td>Saving output</td>
</tr>
<tr>
<td>2.3</td>
<td>Using the Examples menu</td>
</tr>
<tr>
<td>2.4</td>
<td>Operating the program in batch mode</td>
</tr>
<tr>
<td>3</td>
<td>Overview of the Extended Features of Syntax</td>
</tr>
<tr>
<td>3.1</td>
<td>More flexible modeling and parameter restrictions using LG-Equations™</td>
</tr>
<tr>
<td>3.2</td>
<td>Additional models compared to the Cluster, DFactor, Regression, Step3, Markov, and Choice</td>
</tr>
<tr>
<td>3.3</td>
<td>Monte Carlo simulation options</td>
</tr>
<tr>
<td>3.4</td>
<td>Multiple imputation</td>
</tr>
<tr>
<td>3.5</td>
<td>Validation and holdout options</td>
</tr>
<tr>
<td>3.6</td>
<td>Output and saving options</td>
</tr>
<tr>
<td>3.7</td>
<td>Scoring new cases</td>
</tr>
<tr>
<td>3.8</td>
<td>Other options</td>
</tr>
<tr>
<td>4</td>
<td>The Syntax Language</td>
</tr>
<tr>
<td>4.1</td>
<td>Options section</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Maxthreads</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Algorithm</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Startvalues</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Bayes</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Missing</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Montecarlo</td>
</tr>
<tr>
<td>4.1.7</td>
<td>Quadrature</td>
</tr>
<tr>
<td>4.1.8</td>
<td>Step-three options</td>
</tr>
<tr>
<td>4.1.9</td>
<td>Output options</td>
</tr>
<tr>
<td>4.1.10</td>
<td>Outfile options</td>
</tr>
<tr>
<td>4.1.11</td>
<td>An example</td>
</tr>
<tr>
<td>4.2</td>
<td>Variables section</td>
</tr>
<tr>
<td>4.2.1</td>
<td>ID variables</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Weights</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Dependent variables</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Independent variables</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Latent variable information</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Survey options</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Options for Markov models</td>
</tr>
<tr>
<td>4.2.8</td>
<td>Other options</td>
</tr>
<tr>
<td>4.3</td>
<td>Equations section</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Regression equations</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Variances</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Covariances and associations</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Scale factor equations</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Using lists of variables</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Restrictions</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Starting values</td>
</tr>
<tr>
<td>4.3.8</td>
<td>Features for Markov models</td>
</tr>
<tr>
<td>4.3.9</td>
<td>Special parameter coding using the tilde character (~)</td>
</tr>
<tr>
<td>4.3.10</td>
<td>Starting values and saved parameters between ' {{ } } ' or in separate text file</td>
</tr>
<tr>
<td>4.3.11</td>
<td>Unallowed model specifications</td>
</tr>
<tr>
<td>4.4</td>
<td>Selecting a keyword, variable, file, or model</td>
</tr>
<tr>
<td>5</td>
<td>Model Examples</td>
</tr>
<tr>
<td>5.1</td>
<td>Cluster models</td>
</tr>
</tbody>
</table>
5.1.1 Unrestricted cluster model .................................................................51
5.1.2 Restricted cluster model.................................................................52
5.1.3 Cluster model with local dependencies ..............................................53
5.1.4 Cluster model with covariates...........................................................53
5.1.5 Multiple-group cluster model ...........................................................54
5.2 Discrete-factor models and other models with multiple discrete latent variables ....................................................55
5.2.1 Single DFactor model ......................................................................55
5.2.2 Exploratory two-DFactor model .........................................................55
5.2.3 Confirmatory two-DFactor model .......................................................56
5.2.4 Confirmatory two-DFactor with a path model for the DFactors ..........56
5.2.5 Confirmatory two-DFactor model with a path model containing independent variables ..................................................56
5.3 Factor analysis and IRT models ............................................................57
5.3.1 Single factor or IRT model .................................................................57
5.3.2 Confirmatory 2-factor or 2-dimensional IRT model ...............................57
5.3.3 Factor analysis or IRT models with covariates .....................................58
5.3.4 Latent growth models .....................................................................58
5.3.5 Latent growth models as models for a repeated univariate response ....60
5.4 Latent class regression models .............................................................61
5.4.1 Simple LC regression model .............................................................61
5.4.2 LC regression model for two or more dependent variables .................61
5.4.3 LC regression model with a random intercept ...................................62
5.5 Latent Markov models .......................................................................62
5.5.1 Time-homogeneous latent Markov model ...........................................63
5.5.2 Latent Markov model with time-heterogeneous transitions .................64
5.5.3 Latent Markov model with a grouping variable ..................................64
5.5.4 Latent Markov model with a covariate ..............................................64
5.5.5 Mixture (or mixed) latent Markov model .........................................65
5.6 Multilevel LC models .....................................................................65
5.6.1 Simple multilevel LC model .............................................................66
5.6.1 Multilevel LC Model with Covariates ..............................................66
6 Details on Various Syntax Modeling Options ..........................................67
6.1 Alternative regression models for dichotomous and ordinal dependent variables ......................................................67
6.2 Additional regression models for continuous dependent variables ..........68
6.3 Log-linear scale factor models for categorical dependent variables ........68
6.4 Bias-adjusted step-three modeling .................................................69
6.5 Regression models with a cell weight vector (“~wei” option) ..................71
6.6 Continuous-time Markov models ......................................................72
6.7 Knownclass option .......................................................................73
6.8 Using a dynamic latent variable to define an additional level in a multilevel LC model ................................................74
7 Details on Various Syntax Tools and Output Options ...............................76
7.1 Internal parameter structure .............................................................76
7.2 Obtaining additional output and scoring external files .......................80
7.3 Multiple imputation of missing values ...............................................81
7.4 Monte Carlo simulation and power calculation features ......................82
7.5 User-defined Wald tests .................................................................86
7.6 Power computation for Wald tests .....................................................86
7.7 Score tests and EPCs ...................................................................87
7.8 Alternative standard error estimators ..............................................87
7.9 Identification checking ...................................................................88
7.10 Validation and hold-out options ......................................................91
7.11 Options to write output to text files .................................................92
8 Getting Started with LG-Syntax ............................................................94
8.1 The application and the data ............................................................94
8.2 Using a GUI example to setup the Syntax model ..................................95
8.3 Estimating a Syntax model .............................................................101
8.4 Restricting certain effects to be zero or class independent ...................103
8.5 Inclusion of covariates in model ................................................................. 105
References ........................................................................................................... 109
Index .................................................................................................................. 110
1 Introduction

The main goal in the development of LG-Syntax™ was to offer Latent GOLD users additional flexibility. Rather than making the graphical user interface (GUI) more complex by adding new options – such as the ability to specify additional kinds of parameter restrictions, perform simulation studies, and apply the parameters of previously estimated models to score new cases – we decided to build a syntax system in which these kinds of applications and more may be conducted in an open system using an intuitive command language.

This manual describes the LG-Syntax™ module, an add-on to the Advanced version of Latent GOLD release 5.0, which extends the capabilities of Latent GOLD Basic, Advanced, and Choice in several important ways. The accompanying “Upgrade Manual for Latent GOLD 5.0” provides all details regarding the changes between versions 4.5 and 5.0. This manual is organized as follows:

Section 2 describes various menu options that can be used to create, edit, estimate and/or save a syntax model. It also documents how syntax models can be submitted in a single batch run, as opposed to the usual way using the ‘Estimate’ or ‘Estimate All’ Menu commands.

Section 3 provides an overview of the options implemented in LG-Syntax module, and compares it with the Latent GOLD Basic, Advanced, and Choice modules.

Section 4 spells out the full Syntax Language. In this section, the overall structure of a Latent GOLD syntax (.lgs) file is described with detailed documentation for each of the 3 primary components -- Options, Variables, and Equations.

Section 5 provides Model Examples that can be used to learn the basics of the LG-Equations™ language. Specifically, this section illustrates how to specify various types of Cluster, DFactor, IRT, LC regression, multilevel LC, and latent Markov models, as well as other kinds of models. Some of these models can also be specified with the GUI modules, but many cannot. Many more examples can be found (and loaded to be run) in the Examples menu entry.

Section 6 provides more information on various Syntax modeling options. These include alternative models for ordinal and continuous dependent variables, log-linear scale factor models for categorical dependent variables, bias-adjusted step-three modeling, models using cell weights, continuous-time Markov models, and models using knownclass information.

Section 7 provides information on tools and output options which are unique for Syntax, such as saving estimated models with parameters to obtain (additional) output or score other data files, Monte Carlo simulation, hold out cases for validation, multiple imputation, user-defined Wald tests and Score tests, identification checking, alternative complex sampling standard errors, and write output to text files.

Section 8 provides a tutorial style introduction to Syntax. You may wish to read this section first to get a quick introduction to the capabilities.
2 The LG-Syntax User Interface

Before discussing the Syntax language, it is important to note that it is quite easy to get started in a Syntax session by setting up a GUI model in the usual way using the Windows Menu system, and then use the ‘Generate Syntax’ option from the Model Menu. The complete syntax for the model specified will appear automatically in the syntax editor, where it can be modified as desired prior to estimating the model. The GUI model thus serves as the starting point for perhaps a much more elaborate model that cannot be specified in the GUI menu system. Complete details for this are provided in section 2.1.

An alternative way to start to define a syntax model is by looking for an example that is similar to the model you wish to estimate. The Help / Examples menu contains a large variety of examples, illustrating both the simpler models as well as very complex models. See section 8 for an illustration of the use of the Examples menu.

To accommodate batch runs, Latent GOLD 5.0 consists of two executable programs – LG50WIN.exe for normal Windows operation and LG50.exe for batch execution. Syntax models can be specified, structured and estimated using the LG 5.0 Windows Menu system in conjunction with the LG 5.0 syntax command editor. Syntax statements specifying one or more models can be saved in an ASCII file and executed in batch mode at the MS-DOS prompt using LG50.exe. It is also possible to create and edit syntax files using other text editors.

The following subsections provide more details on the Generate Syntax option, the saving of output, the use of the examples from the help menu, and the running of Latent GOLD in batch mode.

2.1 Setting up a Syntax session and saving Syntax models

Since it is quite easy to setup a GUI model in Latent GOLD using the LG5.0 Windows Menu system, it is often useful to begin with a GUI model containing the basic elements of the desired syntax model. This GUI model can then be converted to an initial syntax model automatically using the ‘Generate Syntax’ option from the ‘Model’ menu. The resulting syntax statements can then be modified to specify the desired model using the LG 5.0 syntax editor. The GUI model serving as the starting point may be one that was created earlier and saved as an ‘lgf’ file, or it may be newly created. The syntax editor becomes operational in the Latent GOLD Output Pane upon execution of the ‘Generate Syntax’ command.

To begin with a GUI model saved previously, simply open the previously saved ‘lgf’ file and select ‘Generate Syntax’ from the ‘Model’ menu to generate the initial syntax model. To use the GUI to specify the initial model, open a data file, select one of the standard modules (Cluster, DFactor, or Regression) from the ‘Model’ menu, and specify the desired model and output settings. Following this step you can select ‘Close’ to close the dialogue box, and use the ‘Generate Syntax’ command as described above to generate an initial syntax file. Alternatively, instead of ‘Close’, you can select ‘Estimate’ to estimate the model, prior to generating the syntax.
The ‘Generate Syntax’ command produces a new main tree entry for the syntax session, which contains the syntax statements that are generated for the GUI model. These statements can be edited as desired in the Output Pane which becomes the LG 5.0 syntax editor. For example, Figure 2-1 shows a ‘Syntax Tree’ containing the model ‘3-class’, which was generated from the GUI model of the same name. The GUI model is listed in the ‘GUI Tree’ shown below the ‘Syntax Tree’ in Figure 2-1. The associated syntax statements are shown in the Output Pane, which is organized into 3 primary sections – Options, Variables and Equations.

The ‘Generate Syntax’ command can also be executed directly from the ‘Model’ menu without setting up an initial model, but then only the Options Section appears (with the default options) and the Variables and Equations Section will need to be added by hand.

Figure 2-1: Example of a Syntax Model Generated from a GUI Model

Once the edited version of the model syntax is complete, the model can then be estimated using the ‘Estimate’ option from the ‘Model’ menu. Similar to the estimation of GUI models, upon completion of the estimation, several output file listings are produced in the tree structure. The output sections appear as nested tree entries. As in the GUI models, model summary output becomes visible in the Output Pane by selecting the model name. Similarly, any particular output section becomes visible by simply selecting it (see Figure 2-2). Similar to GUI sessions, multiple syntax models can be estimated with the same data set by editing the copy of the syntax file associated with a new model (named ‘Model2’ in Figure 2-2). Simply modify the syntax statements as desired, and then execute the ‘Estimate’ command.
GUI models are capable of generating a range of models with different numbers of classes (say 1-4 classes) by specifying ‘1-4’ in the Clusters/Classes section of the GUI dialogue box. When the ‘Generate Syntax’ option is used with this type of model setup, separate syntax models are generated for each of the 4 models.
As in GUI models, you can change the name of a syntax model before or after it has been estimated. This is done by selecting the model in the Outline Pane, and then clicking it a second time. You then enter into edit mode as shown in Figure 2-3 above.

Once the syntax has been edited, the ‘Model’ menu command ‘Check Syntax’ can be used to check the syntax statements for consistency. If errors are encountered, messages will appear which display the line numbers within the syntax that are responsible for the errors. The ‘Model’ menu command ‘Estimate’ also performs this check and if no errors are found, estimates the model.

After a syntax model has been estimated, the syntax statements can be viewed in the Output Pane by highlighting the ‘Syntax’ tree entry in the Outline Pane. Since these statements serve to document the model just estimated, they cannot be edited or otherwise modified. However, there are two ways to use this syntax as the starting point to specify a new model. First, if this was the last model estimated, typically an editable syntax for this (last) model appears as a tree entry labeled ‘Model#’ at the bottom of the outline pane immediately following the tree entries for the other model(s). As mentioned above, in this case you may then simply click to highlight ‘Model#’ at the bottom of the tree to display the syntax (‘#’ denotes an integer automatically assigned by the program), edit it as desired and then Estimate the new model.

If no editable syntax currently exists for this model (e.g., you may have deleted it), click to highlight the name of the model, and select ‘Copy Model’ from the ‘Model’ menu. An editable syntax for this model is then produced at the bottom of the tree. The name used for this copy is the same name as the model copied, with the additional characters ‘(1)’ appended. The only exception occurs if the last characters of the model name already is ‘(n)’ for some integer n. In this case, the new name will be the same as this model name with the integer ‘n’ changed to the integer ‘n+1’.

Although you generally will not want to do so, the ‘Copy Model’ option can also be applied to an editable syntax to obtain a second copy of this editable syntax. If applied to the automatically generated model named ‘Model#’, the name of the copy will be ‘Model#+1’, where ‘#+1’ denotes an integer equal to the integer ‘#’ incremented by 1.

Using the Save Syntax Option

The ‘Save Syntax’ default option is similar to the ‘Save Definition’ option used to save a current GUI model setup in a ‘lgf’ (Latent GOLD File) format. The ‘Save Syntax’ option saves a current model setup in a ‘lgs’ (Latent GOLD Syntax) format. The current model to be saved can be either a GUI or a syntax model. Saved ‘lgf’ and ‘lgs’ files can be opened using the File Open command. A ‘lgf’ file is opened in the GUI version of the program, whereas a ‘lgs’ file opens in the syntax editor.

- A current GUI model can be saved as either an ‘lgf’ file and/or an ‘lgs’ file
- A current syntax model can only be saved as an ‘lgs’ file

Since certain syntax models cannot be expressed as GUI models, Syntax models cannot be saved as ‘lgf’ files. When syntax statements are visible in the Output Pane, only the ‘Save Syntax’ option, not the ‘Save Definition’ option, will be accessible from the ‘File’ menu. To save a
current set of syntax statements visible in the Output Pane, select the ‘Save Syntax’ option in the ‘File’ menu and an ‘lgs’ file is created with the name specified.

In addition to the standard default ‘lgs’ format used to save the model settings either before or after a model has been estimated, after a model has been estimated the model parameter estimates may also be saved. See section 0 for a discussion of additional scoring capabilities available when the parameter estimates are saved along with the model settings.

Once the model has been estimated, and the ‘Save Syntax’ option is selected from the ‘File’ menu, the ‘Save Contents’ drop-down box at the bottom of the ‘File Save As’ menu becomes active, and one of the following 3 ‘lgs’ format types may be selected:

1. Syntax: model setup information only (default)
2. Syntax with Parameters: model setup information and parameter estimates (See section 7.1 for details).

Figure 2-4: The drop-down menu in the ‘Save as Type’ box lists these ‘Save as type’ options:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>with Parameters</th>
<th>with Parameters and Variable Definitions</th>
</tr>
</thead>
</table>

2.2 Saving output

The various output Views may be saved using the Save Results command from the File menu. To save only a single entry, highlight it in the Outline Pane and select ‘Save Results’ from the File menu. The selected view (the particular listing highlighted) will then be saved. This will include the Syntax view as well as the Model summary view.

There is also an option to save all output listings associated with a particular model. To do this, select the ‘Model Views’ radial button located under the ‘Save Results’ command in the File menu. This will save all output listings, including the Syntax, in a single output file.

Alternatively, you can save all output listings for all models estimated for a given dataset. To do so, from the File menu select ‘Save Results’, and click the radial button ‘All Views’.

2.3 Using the Examples menu

The interactive ‘Examples’ menu option, accessible through the ‘Help’ menu, can be used to select or create an initial syntax for a particular model which may be modified as desired prior to estimating the model. Separate menus are provided for GUI (‘lgf’) as well as syntax (‘lgs’) examples.

Section 4 of this manual documents the entire syntax language and describes each of the various sections in a syntax file. As you read this and later sections, you may find it useful to refer to the
‘Examples’ menu to retrieve examples for various kinds of syntax models. By default, the ‘Examples’ menu contains several models which can be estimated on our demo data sets. In addition, Tutorial 1 begins by using one of these examples as a starting point and illustrates various changes that can be made in the syntax to specify restrictions and change other settings.

The ‘Examples’ menu is interactive in the sense that model entries may be added or deleted, so that you can customize it to maintain any examples you desire. Once retrieving a syntax example, it can be copied and edited, allowing other models to be estimated on the data. This capability may also be used to document your most recently estimated models by including it as an additional Examples entry. An alternative Examples menu, annotated to illustrate how this type of menu can be developed is available for download from our website at: http://www.statisticalinnovations.com/products/examples.mnu.

You can access syntax examples from the ‘Help’ menu. To navigate to an example of ‘Two Dependent Processes’,

- Select the ‘Syntax Examples’ menu
- Select ‘Latent/Hidden Markov Models’
- Select ‘Two Dependent Processes (1)’

Figure 2-5: The Examples Menu

Figure 2-5 shows the 3 layers in the ‘Examples’ menu. Any model associated with this example may then be open in the syntax editor (see Figure 2-6) and thus be ready to be edited or run.

Figure 2-6 shows the syntax for the ‘Two Dependent Processes (1)’ example. With ‘cross-lagged effects between grade and interest’ selected in the Outline Pane, you may scroll down to view the Equations Section in the Output Pane.

See Section 0: ‘Getting Started with LG-Syntax’ for further instruction regarding the use of the Examples menu.
2.4 Operating the program in batch mode

Latent GOLD 5.0 can run models in "batch mode" via the Windows command prompt. The installation program provides a Start Menu item labeled "Latent GOLD 5.0 Command line" which starts up a Windows command prompt that allows the command "lg50" to be recognized (via the insertion of the application directory into the PATH environment variable).

Models are run by issuing the command:

```
lg50 <model-to-run.lgs>
```

where `<model-to-run.lgs>` is the pathname of a saved Latent GOLD model definition. The results will be placed in a file named `<model-to-run>.lst`. Both GUI (with a .lgf suffix) and Syntax (.lgs) models can be run in this manner.

At present, there is one command line option, "/h": this will cause the output listing to be in HTML style output, instead of a plain text file:

```
lg50 /h model.lgs
```
3 Overview of the Extended Features of Syntax

In this section we provide an overview of the extended features in the LG-Syntax module compared to the Latent GOLD GUI modules. These include the following:

1. More flexible modeling and parameter restrictions by specifying intuitive LG-Equations™
2. Additional models compared to the Cluster, DFactor, Regression, Step3, Markov, and Choice
3. Monte Carlo simulation options
4. Multiple imputation options
5. N-fold validation options
6. Additional output and saving options
7. Options to use saved parameters (e.g., for scoring)
8. Other options

Each of these features is described in more detail below.

3.1 More flexible modeling and parameter restrictions using LG-Equations™

The most important feature is that the model to be estimated is specified using a series of regression equations; that is, regression equations for the latent variables and for the dependent variables (or indicators). These easy to specify regression equations define the linear predictors of the corresponding generalized linear models – linear regression, logistic regression, Poisson regression, etc., depending on the scale type. (Co)variances and associations are also specified as a part of the equations.

An added flexibility through the intuitive LG-Equations language is that certain parameters may be specified in a way that depend on the values of other variables (we call this conditional effects). For example, one possible use of conditional effects is in multiple group analysis, in which some of the parameters in the specified model are assumed to differ across subgroups, say the age effect differs for males and females. This is specified as ‘AGE | SEX’. Also, higher-order interactions may be included easily in a model. For example, inclusion of an interaction between age and gender in a regression model along with main effects for both predictors is simply expressed as ‘AGE + SEX + AGE SEX’.

Labels may be assigned in a natural way to sets of parameters to achieve extended control of the parameters. Using these labels one can specify starting values and parameter restrictions, such as fixed value and monotonicity constraints, and equality and ratio constraints across parameters.
3.2 Additional models compared to the Cluster, DFactor, Regression, Step3, Markov, and Choice

With the syntax module, you can specify many types of additional models compared to the GUI modules. In part, this is the result of the additional flexibility offered by LG-Equations language and the parameter restrictions mentioned above. Other features that make it possible to define additional types of models are:

- A syntax model may contain any combination of nominal, ordinal, and continuous latent variables. This expands upon the GUI modules, which allow the use of either a single *nominal* latent variable (Cluster/Regression/Choice/Markov) or multiple *ordinal* latent variables (DFactor), possibly combined with *continuous* factors (CFactors). With the syntax module, one can specify easily, for example, a model with two nominal latent variables or with one nominal and one ordinal latent variable. Another example is a model with ordinal latent variables at the group level.

- The modeling of continuous latent variables is more general than what is possible in the GUI models. In LG-Syntax models, continuous latent variables can have free variances and be correlated. Also, it is possible to regress continuous latent variables on independent variables, discrete latent variables, and higher-level continuous latent variables. This makes it possible to specify *factor analysis* and *item response* models with covariates, as well as *factor mixture* models in which factor means and (co)variances differ across latent classes, and many other kinds of models.

- Discrete (nominal/ordinal) latent variables can be used in a path model in which one discrete latent variable is used as a predictor in the regression model for another discrete latent variable. Discrete latent variables may also be predicted by continuous latent variables.

- For models with multiple observations per case (models with a case id and repeated measurements for each id) it is possible to specify more than one dependent variable. This, in fact, expands the Regression and Choice modules by allowing for multiple dependent variables. This feature may be used to specify growth models for multivariate responses, as well as other kinds of multivariate regression models. An application in choice modeling is the simultaneous modeling of choice variable with other types of dependent variables (e.g., ratings or counts).

- Latent Markov models, also known as the hidden Markov or latent transition models, can be specified with the new Markov GIO submodule, which is part of Latent GOLD 5.0 Advanced. LG-Syntax 5.0 provides various more extended models with dynamic latent variables, such as latent Markov models with multiple dynamic latent variables, second-order models, and multilevel variants of the latent Markov model (see, for example, Crayen et al. 2012). The dynamic latent variable option can also be used to define multilevel LC models with arbitrary numbers of levels.
A new modeling option in LG Syntax 5.0 is the possibility to define continuous-time latent Markov models. These are latent Markov models that accommodate measurements which are not equally spaced in time.

Latent GOLD 5.0 Basic includes a new Step3 GUI submodule for bias adjusted step-three modeling (and scoring). LG-Syntax 5.0 provides various more extended step-three modeling possibilities, such as step-three models with both covariates and dependent variables and/or multiple latent variables, step-three latent Markov models, and step-three multilevel LC models.

For ordinal dependent variables, in addition to the adjacent-category logit model, in LG-Syntax one can specify models based on cumulative responses probabilities -- cumulative logit, probit, and log-log models -- as well as models for continuation-ratio or sequential logits. The probit and log-log specifications also yield new models for dichotomous responses.

LG-Syntax 5.0 contains new modeling options for continuous dependent variables. In addition to linear regression models with normal error distributions, one can now also use gamma, beta, or von Mises distributions, for non-negative, 0-1 range, and circular variables, respectively.

New in LG Syntax 5.0 is the possibility to specify a log-linear scale factor model for categorical response variables (nominal, all ordinal types, all choice types). Since the scale factor is inversely related to the response uncertainty, this option makes it possible to model heterogeneity in response (un)certainty.

### 3.3 Monte Carlo simulation options

LG-Syntax implements different ways to simulate multiple data sets from a user-specified population model, as well as to run models using simulated data sets. Specification of a population involves defining a file structure and the fixed values for the independent variables in an ‘example’ data file and providing starting values for all parameters in a specified model. Starting values can be provided in different ways, as described in detail in section 4.3.7 of this manual. The different types of simulation options are:

- Multiple simulated data sets can be generated and written to a stacked file containing an additional variable called ‘simulation_#’ which provides a unique identifier for each of these generated data sets. The option ‘simulationid <variable name>’ makes it possible to analyze a data file containing such multiple simulated data sets in a single run. The output will contain summary information, such as parameter means and standard errors across simulation replicates. It is also possible to write the results for all simulation replicates to a compact output file.

- Another simulation option is provided by the ‘MCstudy=<model name>’ option. This, in fact, fully automates the two steps described in option 1 above without saving the data files that are generated. A particular syntax model is run n times, once on each simulation.
data set which is generated from the population defined in ‘<model name>’. Again, summary information is reported as output and detailed results can be saved to a file.

- Monte Carlo simulation can also be used for power calculations using likelihood-ratio tests. This is implemented via the statement ‘power=<model name>’ which is included in the restricted (H0) model. Here, ‘<model name>’ is the name of the unrestricted (H1) population model, from which data sets are generated (with any user-specified sample size). The power procedure in this MC-based power calculation determines for which proportion of the n replicates H0 is rejected in favor of H1 at a 5% significance level (or any user-specified alpha level or critical value).

- Two other Monte Carlo simulation options are the parametric and non-parametric bootstrap procedures. Parametric bootstrap can be used to estimate the p-values of goodness-of-fit and likelihood-ratio tests across nested models. In the LG-Syntax module this feature is available for any type of model. In addition, both types of bootstrap procedures can also be used to estimate standard errors.

Users may control parameter estimation during the Monte Carlo simulations by specifying convergence criteria and maximum number of iterations specific to the Monte Carlo replicates.

### 3.4 Multiple imputation

As a way of handling missing data, the LG-Syntax module can generate multiple imputations based on a simple latent class model (as proposed by Vermunt, Van Ginkel, Van der Ark, and Sijtsma, 2008; see also Van der Palm, Van der Ark, and Vermunt, 2013), or any other model that can be estimated by the program. For example, one may generate multiple imputations using the traditional multivariate normal model. In order to deal with uncertainty with respect to the parameters of the imputation model, we use a non-parametric bootstrap procedure. Similar to the simulation option described above, multiple versions of the imputed data sets are written to a single stacked file containing an extra variable, ‘imputation_#’.

The syntax module cannot only be used for imputing missing values, but it can also be used in the analysis of any data set containing multiple imputations, where the imputations may have originated from a program other than Latent GOLD. Any type of Latent GOLD syntax model can be estimated with a stacked data file containing multiple imputations. To do so simply requires inclusion of the command ‘imputationid <variable name>’ in the syntax file. The program will analyze each of the n data sets separately and combine the results – parameters and standard errors – using the well established procedures for doing so. Moreover, Z and Wald tests are replaced by T and F tests, and some adapted fit information is also provided.

### 3.5 Validation and holdout options

The syntax module has three types of validation procedures that can be used to prevent overfitting. The three procedures are: n-fold validation, holdout cases, and holdout replications.
In *n-fold validation*, cases (or groups in a multilevel analysis) are assigned to one of n validation subsamples. The model of interest is re-estimated n times, where each time one of the n subsamples is excluded. Log-likelihood, classification, and prediction statistics for the nth validation sample are obtained using the parameters of the nth run, the run in which the nth subsample is omitted from the analysis. By summing the statistics over the n subsamples, one obtains the validation log-likelihood, classification, and prediction statistics. This n-fold validation procedure can be performed using either predefined subgroups (using the ‘validationid <varname>’) or the subsamples can be randomly constructed by Latent GOLD.

When *holding out cases*, one portion of the sample (the non-holdout cases) is used for parameter estimation. However, chi-squared, log-likelihood, classification, and prediction statistics are provided for both the estimation sample and the holdout (or validation) sample. Holdout cases are specified using the ‘holdout cases <expression>’ command.

*Holding out replications/responses* for some or all cases is another type of validation method. Parameters are estimated with one set of replications, whereas additional prediction statistics are reported using the other set of the replications/responses for the same cases. Holdout replications are specified using the ‘holdout replications <expression>’ command.

### 3.6 Output and saving options

The syntax module implements various saving and output options which are not available in the GUI modules. Regarding the saving options, it is important to note that after a syntax model is run, the model specification together with the estimated parameters can be saved using the saving option called ‘*Syntax with Parameters*’. When such a saved model is reopened and rerun, iterations will begin again, but use the restored parameters as starting values. This strategy provides an efficient tool for restoring results for models run earlier, as well as a way of requesting additional output sections for a particular model without the need to rerun the model from scratch. This option can also be used when a run did not converge: a new run is thus started from the point where the previous ended. See section 0 for an example.

Various options are available to write output to text files that can be processed with other programs. These include the write/append option yielding the main LG-Syntax output in a comma delimited compact output file, writeexemplarydata yielding a data file containing all possible data patterns with their expected frequencies, writehessian yielding the information matrix, writegradient yielding the gradient contributions for each observation, and writeBCH yielding a data set containing the weights used in adjusted 3-step LC modeling.

It is possible to check the *identification* of models with nominal and ordinal dependent variables. The program determines identification by computing the rank of the Jacobian matrix for a specified number of random values of the model parameters.

There various types of standard errors (S.E.s) that can be obtained for *complex sampling designs* – when a stratumid, psuid, or sampling weight is specified. Not only the Taylor linearization
estimator, but also jackknife, nonparametric bootstrap, and user-defined replicate weights can be used to obtain complex sampling S.E.s. The jackknife and nonparametric bootstrap are also useful for computing S.E.s without complex sampling features. New in LG-Syntax 5.0 is the possibility to obtain S.E.s based on the expected information matrix, possibly approximated by Monte Carlo simulation.

LG-Syntax contains various features for Wald tests. Output from models containing conditional effects (effects that differ across values of another variable, for example, gender) includes an expanded Wald(=) test. Recall that Wald(=) is used in the Regression module to test whether a set of parameters differ across latent classes. In syntax models, the expanded statistic can also be applied to conditional effects, for example to test whether a set of parameters differ across males and females. Moreover, LG-Syntax 5.0 implements an option to obtain user-defined Wald tests for linear contrasts between parameters which should be defined in a separate text file. Also new is the possibility to perform power computations for Wald tests.

New in LG-Syntax 5.0 is the possibility to request score tests for parameter constraints. In conjunction with these score tests, LG will report the expected parameter changes (EPCs) in the restricted set and in the other (free) parameters when a restriction is relaxed; these statistics are called EPC(self) and EPC(other). Strongly related to score tests are bivariate residuals (BVRs). LG-Syntax 5.0 contains various new types of BVRs; that is, conditional, exact, longitudinal, and two-level BVRs. The latter two are also available in particular GUI models.

LG-Syntax generalizes the GUI modules output for the more general types of models available in Syntax. For example, EstimatedValues from the Regression module can be requested in any model, and ProbMeans output reports the values based on the posterior probabilities/means for all latent variables, including group-level latent variables. An alternative version of the ProbMeans output, reporting the marginal estimated class proportion conditional on a single independent variable computed using the model probabilities instead of the posterior probabilities, has also been implemented. For these, we also report standard errors. As in the new Markov module, LG-Syntax provides the Profile-longitudinal output, including its graph.

### 3.7 Scoring new cases

As indicated above, LG-Syntax contains an option to save a model together with its estimated parameters. An alternative option is to save a model with both parameters and additional information on the variables used in the models (number of categories, coding of categories, etc.). The latter is achieved by the saving option ‘Syntax with Parameters and Variable Definitions’.

Saving the variable definitions is useful when applying the results obtained from the analysis data set to other data sets, an application typically referred to as ‘scoring new cases’. Scoring may either involve classifying new cases into latent classes and/or obtaining predictions for the dependent variable(s). The variable definitions are used to identify and control for possible inconsistencies between the analysis data file and the new data file. See section 0 for an example.
An alternative way of classifying new cases into latent classes that will work in most Cluster and DFactor-like models involves computing the scoring equation using the new Step3 option (using either GUI or Syntax) and saving the scoring equation in SPSS or generic syntax format. The scoring can then easily be performed outside Latent GOLD.

3.8 Other options

Other options which are available in Syntax include the following:

- It is possible to define a group-level weight variable in multilevel models.

- The coding options for nominal variables are more expanded than in GUI model. In addition to effect coding, and first and last reference categories being selected for dummy coding, it is possible to select any category as the reference category for dummy coding. In addition, it is possible to specify a different coding for each nominal variable.

- There are various special types of coding options for parameter sets. These are referred to as nominal, ordinal, squared, difference, full, transition, error coding, weight, intercept, and missingvaluedummy; that is,
  - nominal: nominal coding for an ordinal dependent/latent variable
  - ordinal: ordinal scores coding for a nominal dependent/latent variable
  - squared: squared scores for an ordinal dependent/latent variable
  - difference: coding as differences between adjacent categories for nominal variables
  - full: coding without identifying constraints for nominal variables
  - transitions: coding of parameters in Markov models as the logit of transitions
  - error: coding the parameters as the logit of measurement error, applicable when nominal latent and nominal dependent variables have the same number of categories
  - weight: defines a vector with cell weights, which are basically fixed parameters in exponentiated form
  - intercept: indicates that the effect concerned is actually the intercept term
  - missingvaluedummy: generates a term consisting of a missing value dummy for the independent variable concerned

- The known-class option, used in GUI models to prevent selected cases from being assigned to particular latent classes of a single latent variable, may be used in the syntax more than once in models containing multiple latent variables.

- An exposure variable or a fixed exposure constant may be used with any dependent variable which is specified as a Poisson or binomial count.

- The number of nodes used in the numerical integration of continuous latent variables can take on any number between 2 and 100, and can differ for different latent variables. In
addition, user-defined nodes and weights can be provided, which makes it possible to utilize distributions other than normal.

- Two other Newton-type algorithms are implemented: the Berndt-Hall-Hall-Hausman (BHHH) and the Levenberg-Marquardt (LM) algorithm.

### 4 The Syntax Language

The following describes the overall organizational structure of a .lgs syntax file; that is, the format that Latent GOLD uses when it saves possibly multiple model definitions of a syntax session as a file with extension ‘.lgs’.

```
//LG 5.0//
version = 5.0
infile '<filename>' '<file format options>'

model
<definition of 1\textsuperscript{st} model>
end model

model
<definition of 2\textsuperscript{nd} model>
end model

model
<definition of m\textsuperscript{th} model>
end model

model
<definition of last model>
end model
```

Users may also construct their own file with such a structure, or modify a saved syntax file, for example, by adding new models.

When (re)loading a ‘.lgs’ file using the ‘Open’ command from the ‘File’ menu, the name of the data file appears as the main entry and <definition of 1\textsuperscript{st} model>, <definition of 2\textsuperscript{nd} model>, etc. appear as subentries. Note that this overall structure, containing a statement for the file name definition and ‘model’ and ‘end model’ statements surrounding the various models, is NOT visible within the Latent GOLD syntax editor to reduce the possibility of errors introduced during the editing process. (See below for a description of the portion that \textit{is} visible within the syntax editor.)

The name of the data file is provided in the 3rd line of the .lgs file:
infile ‘<filename>’ ‘<file format options>’

Here, ‘<file format options>’ refers to the separation and the quoting used in text format data files.

When selecting a particular model, the statements defining the model are organized into the four sections: ‘title’, ‘options’, ‘variables’, and ‘equations’. The Title Section is optional. If present, the title given in the ‘title’ section is displayed as the name in the syntax tree in the Outline Pane. The other three sections are visible in the Latent GOLD syntax editor. The Title Section (if present), must appear first, followed by the Options and Variables Sections (in either order), followed by the Equations Section. If parameter restrictions are included, they must appear in the bottom portion of the Equations Section.

```plaintext
title ‘<title>’;
options
    <definition of technical and output options>;
variables
    <definition of variables that are read from the data file and of the latent variables included in the model>;
equations
    <regression, variance and association/covariance equations defining the model, as well as restrictions and starting values for parameters>;
```

The Title Section ‘title’ can be used to change the default model name (Model1, Model2, etc.) to a more meaningful user-defined name. This section consists of a single statement; that is,

```plaintext
title ‘<title>’;
```

Example:

```plaintext
title ‘3 by 2 cluster model’;
```

Default settings are used for the ‘options’ and ‘equations’ sections if omitted.

Comments can be included in any portion of the syntax. Comments consist of text between ‘//’ and the end of line. Also, for extensive comments that consist of multiple lines, such text may be included between ‘/*’ and ‘*/’. Comments are ignored when evaluating the syntax file.

A special type of ‘lgs’ file consists of the above sections plus Parameter Values (following the Equations Section). When this special type of ‘lgs’ file is used the command ‘Estimate’ causes the parameter values to be used as the initial estimates. This special type of ‘lgs’ file is produced as an option in the Save Syntax command (see section 2.1) and may be used for different purposes including scoring additional files (see section 0). When parameter values from previously estimated models are included, an additional section containing Variable Definitions may also be present (at the beginning of the file) to assist in the scoring of other data files. (See the Save as Type option ‘Syntax with Parameters and Variable Definitions’ in section 2.1 for details).
4.1 **Options section**

This section defines the technical and output options that correspond to the Technical and Output tab GUI options. The complete list of options – using %f to denote a real number, %d to denote an integer number, and bold face to denote the default setting – is:

- maxthreads=<all|d>:

- algorithm
  - <emtolerance=%f>
  - <tolerance=%f>
  - <emiterations=%d>
  - <nriterations=%d>
  - <infmat=hessian | LM | BHHH | expected | numerical>
  - <MstepNR>;

- startvalues
  - <sets=%d>
  - <iterations=%d>
  - <seed=%d>
  - <tolerance=%d>;

- bayes
  - <latent=%f>
  - <categorical=%f>
  - <poisson=%f>
  - <variances=%f>;

- missing
  - <excludeall | includedependent | includeall>;

- montecarlo
  - <L2 | allchi2 | LLdiff | power | MCstudy [= 'model name' | %d]>
  - <seed=%d>
  - <replicates=%d>
  - <N=%f>
  - <alpha=%f>
  - <CV=%f>
  - <sets=%d>
  - <emtolerance=%f>
  - <tolerance=%f>
  - <emiterations=%d>
  - <nriterations=%d>
  - <iterationdetails>
  - <nowrite>
- quadrature
  <nodes=%d>;

- step3 [proportional | modal | ml | bch | none] [simultaneous]:

- output
  <parameters [= effect | last | first]>
  <profile [= longitudinal | posterior]>
  <probmeans [= model | posterior]>
  <bvr [= marginal | conditional | exact]>
  <bvrlongitudinal>
  <bvrtwolevel>
  <frequencies>
  <classification [= model | posterior | forward]>
  <estimatedvalues [= model | regression]>
  <iterationdetail(s)>
  <standarderrors [= standard | robust | fast | none | jackknife | npbootstrap [%d]
  bootstrap [%d] | expected | MCexpected | numerical >
  <identification [= %d]>
  <validationLL [= %d]>
  <predictionstatistics [= model | posterior | hblike]>
  <write | append = 'filename'>
  <samplesizeBIC=%f>
  <ParameterCovariances [= 'filename']>
  <ParameterCorrelations [= 'filename']>
  <ignoreknownclass>
  <waldtest='filename'>
  <waldpower=%f>
  <scoretest>
  <writehessian='filename'>
  <writegradients='filename'>
  <writeparameters='filename'>
  <writeexemplary='filename'>
  <writeBCH='filename'>
  <writeclassificationerrors='filename'>
  <writeclassificationerrorsproportional='filename'>
  <writeSPSSsyntax='filename'>
  <writegensyntax='filename'>;

- outfile '<filename>'
  <classification [= model | posterior | forward]>
  <prediction [= model | posterior | hblike]>
  <individualcoefficients>
  <simulation [= %d] [seed=%d]>
  <imputation [= %d] [em] [seed=%d]>
  <cooksD>
Note that each statement contains one or more options, and ends with a ‘;’. When statements or options are omitted, the program’s default settings apply.

4.1.1 Maxthreads

The technical option maxthreads is used to indicate the maximum number of parallel processes (called threads) LG should set up during parameter estimation. The default is ‘all’ indicating that this number should equal the number of cores of the computer used for the computations.

4.1.2 Algorithm

The algorithm subsection defines the four technical settings referred to as ‘Convergence Limits’ and ‘Iteration Limits’ in GUI models. Two additional options are 1) set the type of information matrix to use in the Newton algorithm (inmat = hessian | BHHH | LM | expected | numerical), and 2) to request the use of Newton type of iterations within the M step of the EM algorithm (MstepNR).

The option hessian yields the standard Newton-Raphson algorithm which uses the observed information matrix (or the negative Hessian) and which is the default. BHHH yields the ‘quasi-Newton’ Berndt-Hall-Hall-Hausman algorithm which approximates the information matrix using the outer-product of the first derivatives. LM yields the Levenberg-Marquardt algorithm which is a stabilized version of the BHHH algorithm. The option expected yields the Fisher scoring algorithm, which uses the expected information. The option numerical yields a standard Newton-Raphson algorithm, but with the hessian (matrix of second derivatives of the log-likelihood) computed numerically.

4.1.3 Startvalues

This subsection defines the settings for the multiple random ‘Starting Values’ procedure. These options are the same as in GUI models; that is number of sets, number of EM iterations per set, and the convergence criterion.

The procedure to generate the random starting values in the LG-Syntax is somewhat different than that in the GUI. Thus, a seed used in a GUI model will not necessarily obtain the same model in the LG-Syntax.

4.1.4 Bayes

This subsection allows the utilization of priors to prevent boundary solutions. It corresponds to the ‘Bayes Constants’ on the Technical tab on GUI models.
The default values for the four bayes constants are now 0, whereas in GUI models the
defaults are equal to 1. However, if the ‘Generate syntax’ is used to generate the syntax from a
GUI model having the bayes default of 1 (or any other value) that bayes value will be retained in
the syntax.

4.1.5 Missing

This subsection defines how to deal with missing values. As in the GUI ‘Missing Values’
settings, it is possible to (1) exclude all records with missing values, (2) include records with
missing values on the dependent variable(s) (but exclude the ones with missing values on
independent variables), and (3) include all records with missing values in the analysis.

Missing values in Markov models: In Markov models, it is assumed that the records of
an individual refer to equally-spaced time points. Therefore, for Markov models it is important
not to exclude cases with missing values, because otherwise the time structure of the data set will
be distorted. For Markov models, you should typically use the setting ‘missing includeall’.

4.1.6 Montecarlo

The ‘montecarlo’ subsection defines the settings for the simulation and resampling methods
implemented in LG-Syntax, which are

- parametric bootstrap for chi/squared, BVR, likelihood-ratio test p values and S.E.s
- non-parametric bootstrap S.E.s
- replicate weights and jackknife based S.E.s
- Monte Carlo expected information matrix based S.E.s
- validation log-likelihood
- Monte Carlo simulation studies
- power calculation by simulation
- simulation of data sets
- multiple imputation

There are three types of ‘montecarlo’ settings: (1) settings that concern all simulation and
resampling procedures, (2) settings that concern specific procedures, and (3) setting for
requesting specific Monte Carlo simulation based output; that is,

- The options for requesting a particular type of MC simulation feature are L2, allchi2,
  LLdiff, power, and MCstudy. L2, allchi2, and LLdiff are extensions of the parametric
  bootstrap procedure implemented in the GUI modules. L2 yields a bootstrap p value for
  the L-squared goodness-of-fit statistic, allchi2 yields bootstrap p value for all goodness-
  of-fit statistics including the BVRs, and LLdiff='model name' (or %d) a bootstrap p value
  for the likelihood ratio test comparing the current model with the model called 'model
  name' (or with number %d). The option power='model name' (or %d) can be used to
calculated the power of a likelihood-ratio test comparing the current (restricted) model
with another (unrestricted) model called 'model name'. The MCstudy option runs a
simulation study in which the current model is run with either data simulated from the current model, or with data simulated from another model called 'model name'.

- With <replicates=%d>, one sets the number of replicates in parametric and non-parametric bootstrap, in MC studies and in power calculations by MC simulation.
- The option '<seed=%d>' defines the seed that should be used in the generation of simulated/resampled data sets. This makes it possible to replicate previous results.
- The option <N=%f> sets the sample size for MC studies and power calculation, <alpha=%f> sets the type I error level for power calculations and defines the critical value that is requested in bootstrap and MC studies, and <CV=%f> sets the critical value for power calculations (instead of the alpha level).
- The option <sets=%d> can be used to request random start sets for each montecarlo replicate. By default the estimates from the sample (in bootstrap) or the specified population values (in MCstudy and power) are used, except for the non-population model in MCstudy and power, for which random start sets are used when no user-specified start values are provided.
- The options <emtolerance=%f>, <tolerance=%f>, <emiterations=%d>, <nriterations=%d> are the same as in the ‘algorithm’ subsection, and can be used to influence the estimation algorithm use within the simulation/replicate runs. When omitted, the program will use the settings specified in ‘algorithm’ subsection.
- The iterationdetails option is to obtain iterationdetails for the montecarlo replicates themselves; for example, to see the iterations of each bootstrap sample.
- The option nowrite can be used to suppress the writing of the output for each replicate sample to the write/append compact output file. Only the final results will be written to this file.

⚠️ In power and MCstudy, the model from which data are simulated should contain starting values for all model parameters since these will serve as the population values. The other (calling) model may also contain starting values to speed up estimation, but this is not required.

4.1.7 Quadrature

This option is used to set the number of quadrature points for the numerical integration of continuous latent variables. It is the same as in GUI models, except that any number can be used between 2 and 100 (default = 10). In the Variables Section, it is possible to define the number of quadrature points for each continuous latent variable separately, which overrides the overall setting in ‘quadrature’.

4.1.8 Step-three options

With the step3 command one defines the setting for the step-three modeling. These options concern:

- The class assignment method, which can be proportional or modal, with proportional as default.
- The adjustment method, which can be ml, bch, or none, yielding the maximum likelihood method with fixed classification errors, the weighted analysis method proposed by Bolck,
Croon, and Hagenaars (2004), and an unadjusted analysis, respectively. By default LG-Syntax uses ml.

- Whether the model should be estimated in pieces or simultaneously. By default first the model for the latent variables is estimated, then the model for the first dependent variable, then for the second dependent variable, etc. With the keyword ‘simultaneous’ one overrides this setting, implying that all parameter will be estimated simultaneously (as in other LG-Syntax models). The simultaneous option is useful, for example, if one wishes to estimate a step-three model with both covariates and a dependent variable. It can also be used if one wishes to use the standard random starting values procedure, rather than the non-random starting values of the step-three analysis procedure.

In models in which one or more id variables needs to specified, only the modal-ml and modal-none combinations can be used. This applies to mixture regression models with replications, latent Markov models, and multilevel LC model.

4.1.9 Output options

LG-Syntax provides the same output sections as the GUI models, some of which are generalized to deal with multiple latent variables, multiple dependent variables, and the dynamic (Markov-type) latent variables. These sections are:
- Statistics, which includes classification tables for all discrete latent variables and prediction statistics for all dependent variables
- Parameters, which now contains the internal parameter number, the parameter label and index, a Wald(=) test for all types of conditional effects, and separate section ‘(co)variances continuous latent variables’ needed when numerical integration (and thus Cholesky factorization) is used.
- Profile, which combines the GUI Profile (including joint Profile from DFactor models) and GProfile output, and contains additional information in Markov models (dynamic latent class probabilities given case- and group-level latent classes, and marginal transition probabilities). Contrary to GUI models, it does not provide information for covariates. New in version 5.0 is that Profile also contains an interactive plot.
- Profile-Posterior gives a version of Profile obtained using the posterior membership probabilities. This output section also contains information on covariates.
- Profile-longitudinal is a version of Profile for longitudinal data with fixed measurement occasions (e.g. latent Markov or LC growth models). It gives the same information as Profile, but conditioned on time. This makes it possible to see the estimated (per class/state and overall) and observed time trends of the dependent variables. Profile-longitudinal contains an interactive plot.
- Probmeans (called Probmeans-Posterior), which also yields average posterior probabilities/means also for group-level and dynamic (Markov-type) latent variables
- Frequencies.
- Bivariate Residuals, including the new longitudinal and two-level BVRs.
- Classification (called Classification-Posterior).
- Classification-Model (classification based on the independent variables only).
- EstimatedValues-Regression, which yields the same output as in the Regression module, but now for each dependent variable and generalized to multiple discrete latent variables at the lower level.
- EstimatedValues-Model, containing the estimated model probabilities and means for the discrete latent variables and the dependent variables, evaluated at the value 0 of the continuous latent variables.
- IterationDetails.
- Writing the scoring equation to file, either as SPSS syntax language or in a more generic form.

Output sections specific for Syntax are:
- Probmeans-Model, which contains the class proportions for discrete latent variables, and for continuous latent variables means conditional on independent variables, which are based on the estimated model probabilities/means for the latent variables (rather than the posterior probabilities/means).
- Classification-Forward is a special classification output type for Markov models, yielding classification probabilities for dynamic latent variables based on all observed information before the time point concerned (the current class membership is predicted from the past information only).
- ParameterCovariances is a keyword that requests the Parameter Covariance matrix to be displayed within the Parameters output section. If the optional file name is specified after this keyword (=filename'), the matrix will instead be written as a tab-delimited text file. The keyword can also appear in both forms (with and without the file name), in which case it will appear both in the listing as well as the output file.
- ParameterCorrelations (and optional =filename') requests the Parameters Correlation matrix instead of the Parameters Covariance matrix.
- EPC(other) contains the expected parameter changes for the (other) free parameters, as well as the score tests. This output is requested with the option ‘scoretest’.

Additional output options include the possibility
- to request model identification check, where the rank of the Jacobian matrix is computed for the specified number of random parameter sets (default is 10),
- to obtain score tests and EPCs (requested with the scoretest option), appearing in Parameters,
- to compute the power or the required sample size for the Wald tests when using the waldpower option, appearing in Parameters,
- to compute an n-fold validation log-likelihood, where the specified number is the number of folds and where the n-folds are generate randomly,
- to modify the sample size to be used in the BIC computations,
- to request jackknife and (non)parametric bootstrap based standard errors, standard errors based on the expected information matrix, and standard errors based on the observed information matrix which is computed numerically,
- to request exact and conditional BVRs for categorical response variables,
- to write or append statistics, parameters, profile, probmeans= model, BVRs, and estimatedvalues= model to a compact comma delimited text file that can be processed by another program (in the simulation and resampling procedures this is done for each replicate sample),
- to write to a text file the (inverse) information matrix (writehessian), the gradient contributions (writegradient), all data patterns with their expected frequencies (writeexemplarydata), the parameters (writeparameters), the weights used in the 3-step bch adjustment (writebch), and the classification error logits needed for the 3-step ml adjustment (writeclassificationerrors and writeclassificationerrorsproportional).

Note that the coding for nominal variables is defined using the ‘[= effect | last | first]’ option of ‘parameters’. This coding can be overwritten by specifying the coding for a particular variable in the Variables Section.

Standard errors should be requested explicitly using the option ‘standarderrors’. This yields standard errors for parameters, profile, estimatedvalues=model, and probmeans=model. The options standard (default), robust, and fast are also available in GUI models. The additional options are jackknife, non-parametric bootstrap (npbootstrap), and parametric bootstrap (pbootstrap), where the ‘[%d]’ indicates the number of replicates to be used. If the latter number is not specified, the number of replicates specified in the montecarlo statement is taken. It is also possible to obtain S.E.s based on the (monte carlo) expected information. It should be noted that jackknife and npbootstrap can also be used in combination with complex sampling options, yielding complex sampling version of the jackknife (leaving out psu’s one by one) and the nonparametric bootstrap (resampling psu’s within strata). For more detail, see section 7.8.

In Syntax it is possible to request ‘exact’ and ‘conditional’ bivariate residuals for nominal and ordinal dependent variables. The exact BVRs are chi-squared distributed (are in fact Score tests). Conditional BVRs are computed within classes and the aggregated over the classes (see Tay, Vermunt and Wang, 2013). The options BVRlongitudinal and BVRtwolevel can be used to request the new types for BVRs for two-level and longitudinal data sets, which are also available in LG Advanced.

In Syntax, it possible to obtain joint classification information; that is, classification probabilities for various latent variables simultaneously.

Prediction statistics should be requested explicitly, where ‘[= model | posterior | hblike]’ defines the three prediction types which are also available in the Regression module.

The identification check is implemented for models containing only categorical dependent variables. This identification check makes it possible to distinguish more specifically between non-identified parameters and boundary solutions. Moreover, this check will also detect non-identification when bayes constants are used. For more detail, see section 7.9.

When using the option ‘ignoreknownclass’, classification statistics and classification-posterior will be obtained without using known-class information.

Output sections that are not available in Syntax are the factor-like output (loadings, correlations, error correlations) from Cluster and DFactor and the random-effects output from Regression.
 Various information can be output to a file by simply using the keywords ‘outfile’ to specify the output filename followed by keywords identifying the type of output. For example as in the GUI modules, it is possible to output classification and prediction information to an external data file. This is achieved with the following ‘outfile’ command,

- **outfile 'd:\classpred.sav' prediction classification**;

The options ‘[= model | **posterior** | forward]’ and ‘[=model | **posterior** | hblike]’ are the same as discussed above for classification output and prediction statistics. The option classification = forward is a new option for Markov models. Note that classification = model and classification = posterior are also referred to as **covariate** and **standard classification**, respectively.

As in the GUI Regression module, **prediction** yields predicted values for the dependent variable(s) and **individualcoefficients** yields posterior means and standard deviations for the parameters in the regression model(s) for the dependent variable(s). Both output options have been generalized to work with any syntax model. Moreover, **individualcoefficients** now also include the variation associated with continuous latent variables.

Specific for Syntax is the option for simulating data sets from a user-defined population model (option ‘simulation’), where ‘[=%d]’ indicates how many data sets should be generated (default is 1) and ‘seed=%d’ is the seed to be used for the simulations (default is a random seed). The multiple data sets will be stacked into a single data set, with a new variable ‘simulation_#’ indicating the data set number. Such a data set can be analyzed with LG-Syntax using the ‘simulationid’ command in the ‘variables’ section. Simulation requires an example data set defining the values of the independent variables, as well the multilevel or repeated measures file structure (the id variables). Group, case, and replication weights can be used to indicate the number of records that should be simulated. For more detail, see section 7.4.

Another option specific for Syntax is the possibility to perform multiple imputation (option ‘imputation’). Missing values are replaced by random draws based on the estimated model. With ‘[=%d]’ one indicates how many complete data sets should be created (default is 1) and with ‘seed=%d’ one defines the seed to be used for the imputations (default is a random seed). The multiply imputed data sets will be stacked into a single data set, where a new variable ‘imputation_#’ indicates the data set number. Such a data set can be analyzed with LG-Syntax using the ‘imputationid’ command in the ‘variables’ section. Parameter uncertainly is dealt with using a non-parametric bootstrap procedure: each imputed data set is obtained with a new set of parameter estimates from a resample of the original data file. Instead of using this nonparametric bootstrap procedure, one can also request imputations using the ML estimates, which is achieved with the option ‘em’ (referring to what is typically called EM imputation). For more detail, see section 7.3.

The new option ‘keep’ makes it possible to create an output file not only containing the classification/prediction information and the variables used in the analysis, but also any of the
other variables in the analysis (input) file. This means that it is no longer necessary to merge the Latent GOLD classification/prediction output file with the original file prior to performing subsequent analyses, for example, a step-three analysis.

4.1.11 An example

An example of an ‘options’ section is:

options
    maxthreads=4;
    algorithm emtolerance=0.01 tolerance=1e-008 emiterations=250 nriterations=50;
    startvalues sets=100 iterations=250 seed=0 tolerance=1e-005;
    bayes latent=1 categorical=1 poisson=1 variances=1;
    missing includeall;
    output parameters=first standarderrors estimatedvalues probmeans profile bvr
        iterationdetails identification;
    outfile 'classif.sav' classification
        keep age gender;

4.2 Variables section

This section defines the variables to be read from the data file and specifies the latent variables to be included in the model. Each subsection below describes the variable identifiers associated with one of the following six categories: ID variables, weights, observed variables used in the model, latent variables, survey options, and some other types of variables. It may be helpful when editing this section of your LG-Syntax to retrieve the variable names from your data file. You may right-click on the LG-Syntax window and select ‘Variables’. Once the list opens, you may select any variable name to have it written to the desired location in the syntax file.

4.2.1 ID variables

- timeid <varname>;
- caseid <varname>;
- groupid <varname>;

The ‘caseid’ variable is used when there are multiple replications per case, as in mixture regression models. ‘Groupid’ is used in multilevel models to indicate which cases belong to the same group. Note: caseid is a level-2 ID and groupid is a level-3 ID.

In latent Markov models, multiple replications identify the multiple time points associated with a particular case, and caseid associates these time point records with the appropriate case. Timeid is used when there are multiple replications per time point in latent Markov models. In such models, timeid would be the level-2 ID, caseid the level-3 ID, and groupid, when present, the level-4 ID.
4.2.2 Weights

- replicationweight <varname>;
- caseweight <varname>;
- groupweight <varname>;

Frequency weights may be applied at the replication, case, and/or group level. Note that the ‘groupweight’ option is specific to LG-Syntax. It is not available in GUI models. A ‘replicationweight’ applies to a single record, a ‘caseweight’ to a case (records joined with a caseid), and a ‘groupweight’ to a group (records joined with a groupid).

4.2.3 Dependent variables

- dependent <varname>
  <nominal | ordinal | continuous | poisson | binomial | cumlogit | probit | loglog1 | loglog2 | seqlogit1 | seqlogit2 | gamma | beta | vonmises >
  <nominal | ordinal | continuous | poisson | binomial | cumlogit | probit | loglog1 | loglog2 | seqlogit1 | seqlogit2 | gamma | beta | vonmises > is used to define the type of regression model (error and link function) used for the dependent variable: multinomial logistic (nominal), adjacent-category ordinal logistic (ordinal), linear normal (continuous), log-linear Poisson (poisson), binary logistic for counts (binomial), cumulative ordinal logistic (cumlogit), ordinal probit (probit), ordinal log-log (loglog1 or loglog2), sequential logit (seqlogit1 or seqlogit2), log-linear gamma (gamma), logit beta (beta), and linear von Mises (vonmises). Sections 6.1 and 6.2 provide information on the various types of models for ordinal and continuous variables which are only available in Syntax.

The option ‘overdispersed’ can be combined with Poisson and binomial counts, yielding negative-binomial and beta-binomial models, respectively. The ‘censored’ keyword can be used with continuous dependent variables, yielding tobit regression models, and ‘truncated’ with continuous, Poisson, and binomial, yielding zero-truncated versions of the corresponding regression models. ‘Zeroinflated’ can be used with any scale type, but only for the last dependent
variable in the list and for models containing a case level discrete latent variable. ‘Overdispersed’ can be combined with ‘truncated’ or ‘zeroinflated’, but ‘truncated’ and ‘zeroinflated’ cannot be combined.

The complete list of regression models is:

- nominal: binary or multinomial logistic
- ordinal: binary or adjacent-category ordinal logistic
- continuous: normal linear
- continuous censored: tobit (or censored normal) linear
- continuous truncated: zero-truncated normal linear
- Poisson: log-linear Poisson
- Poisson truncated: log-linear zero-truncated Poisson
- Poisson overdispersed: log-linear negative-binomial
- Poisson overdispersed truncated: log-linear zero-truncated negative-binomial
- binomial: binary logistic
- binomial truncated: zero-truncated binary logistic
- binomial overdispersed: logistic beta-binomial
- binomial overdispersed truncated: logistic zero-truncated beta-binomial
- cumlogit: binary or cumulative ordinal logistic
- probit: binary or ordinal probit
- loglog1 and loglog2: binary or ordinal negative/complementary log-log
- seqlogit1 and seqlogit2: binary or sequential (continuation-ratio) ordinal logistic
- gamma: log-linear gamma model for nonnegative continuous variables
- beta: logit beta model for continuous variables with values between 0 and 1
- vonmises: linear von Mises model for circular or directional data

Except for the truncated types, each of these can also be defined as a ‘zeroinflated’ variant. For continuous, Poisson, and binomial, the ‘zeroinflated’ option automatically adds an additional latent class in which the response is always 0 (i.e., the response is 0 with probability 1). For the other scale types, one class is added for each response category, where each added class gives one of the responses with probability 1.

The option `<%d>` can be used to specify the number of categories of a discrete dependent variable. Usually there is no need to use this option because the program determines this number from the data file. An exception is the situation in which the number of possible response categories in the population is larger than the number of categories used by the sample respondents, as observed in the data file. Another situation in which one may use this option is in simulation studies (see section 7.4).

The ‘exposure’ keyword, can be used to specify the value or the variable containing the exposure in Poisson models or the total in binomial models. If omitted, the default exposure value used for Poisson counts is 1, while for binomial counts the default is the largest observed response in the data file.
The options ‘rank’ and ‘score’ implement the GUI counterparts to the Latent GOLD ‘grouping’ and ‘rescoring’ options. Finally, ‘coding’ can be used to change the coding of a nominal dependent or independent variable, which overrides the global setting in the ‘options’ section of the syntax file. Note that for dummy coding, it is possible to specify not only the first or the last but also any other category as reference category.

The option ‘ignoreclassification’ is used to obtain classification information ignoring the dependent variable concerned. The posterior classification information will then be based on the independent variables and the non-ignored dependent variables.

Example:

dependent y1 nominal coding=2, y2 ordinal score=(1,2,4), y3 poisson overdispersed exposure=exp3, y4 continuous truncated;

4.2.4 Independent variables

- independent <varname> <nominal | numeric> <inactive> <coding= effects | first | last | %d> <rank=%d> <score=(%f, %f,…, %f)>, next independent variable, … ;

This option can be used to define the independent variable(s) in the model. Note that independent is a generic term; that is, it includes variables identified in the GUI models’ as predictors and covariates. Other terms are concomitant variables, explanatory variables, external variables, exogenous variables, etc. In any given model, a variable cannot be specified as both dependent and independent.

For independent variables, the scale type can be defined to either numeric or nominal. In addition, as in GUI models it is possible to indicate that independent variables are inactive. The other three options (coding, rank, and score) are the same for independent variables as for dependent variables.

Example:

independent gender nominal, education, age rank=4, region nominal inactive;

4.2.5 Latent variable information

These are the keywords to define the latent variables

- latent <latvarname> <nominal | ordinal | continuous>
The command ‘latent’ identifies the latent variables that are part of the model. The keyword

\texttt{<nominal | ordinal | continuous>}

defines the scale type of the latent variable concerned, which can be nominal, ordinal, or continuous.

For settings ‘nominal’ or ‘ordinal’, the scale type is followed by a nonnegative integer (\texttt{<\text{d}>}) representing the number or classes (levels) for the latent variable concerned. If the number of classes (levels) is not specified, the default value of 2 is used. When numerical integration is needed for a ‘continuous’ latent variable, \texttt{<\text{d}>} can be used to overwrite the number of quadrature nodes defined in the option section (or the default of 10 nodes).

Whether a latent variable is group-, case-, or time-specific is indicated with

\texttt{<group | case | dynamic>}

By default, latent variables are case specific. A group-specific latent variable is so designated using the ‘group’ keyword while a time-specific latent variable is so designated using the keyword ‘dynamic’ (optionally, the keyword ‘markov’ may be used in place of the keyword ‘dynamic’). Group-specific latent variables are needed in multilevel latent variable models. (Dynamic latent variables are needed in latent or hidden Markov models. The keyword ‘dynamic’ can also be used to define latent variable models with an additional (lower) level, or with some tricks) actually any number of additional (lower) levels.

For continuous latent variables, it is possible to define a distribution other than normal by specifying the nodes and weights to be used in the numerical integration instead of the Gauss-Hermite nodes and weights. This is achieved with the commands

\texttt{<nodes =(<\text{f}, <\text{f},..., <\text{f}>)>}

and

\texttt{<distribution=(<\text{f}, <\text{f},..., <\text{f}>)>}

next latent variable, …;

The command ‘latent’ identifies the latent variables that are part of the model. The keyword

\texttt{<nominal | ordinal | continuous>}

defines the scale type of the latent variable concerned, which can be nominal, ordinal, or continuous.

For settings ‘nominal’ or ‘ordinal’, the scale type is followed by a nonnegative integer (\texttt{<\text{d}>}) representing the number or classes (levels) for the latent variable concerned. If the number of classes (levels) is not specified, the default value of 2 is used. When numerical integration is needed for a ‘continuous’ latent variable, \texttt{<\text{d}>} can be used to overwrite the number of quadrature nodes defined in the option section (or the default of 10 nodes).

Whether a latent variable is group-, case-, or time-specific is indicated with

\texttt{<group | case | dynamic>}

By default, latent variables are case specific. A group-specific latent variable is so designated using the ‘group’ keyword while a time-specific latent variable is so designated using the keyword ‘dynamic’ (optionally, the keyword ‘markov’ may be used in place of the keyword ‘dynamic’). Group-specific latent variables are needed in multilevel latent variable models. (Dynamic latent variables are needed in latent or hidden Markov models. The keyword ‘dynamic’ can also be used to define latent variable models with an additional (lower) level, or with some tricks) actually any number of additional (lower) levels.

For continuous latent variables, it is possible to define a distribution other than normal by specifying the nodes and weights to be used in the numerical integration instead of the Gauss-Hermite nodes and weights. This is achieved with the commands

\texttt{<nodes =(<\text{f}, <\text{f},..., <\text{f}>)>}

and

\texttt{<distribution=(<\text{f}, <\text{f},..., <\text{f}>)>}

next latent variable, …;
The options \(<\text{scores}=(%f, %f, \ldots, %f)\) for ordinal latent variables and \(<\text{coding}=<\text{effects} \mid \text{first} \mid \text{last} \mid %d>\) for ordinal and nominal latent variables have the same functionality as the same option in dependent and independent.

The ‘knownclass’ option can be used to include partial classification information for one or more nominal/ordinal standard latent variables. In applications where a subset of the cases are known with certainty not to belong to a particular class, or particular classes, a knownclass indicator can be used to restrict their posterior membership probability to 0 for specified classes and hence classify these cases into one of the remaining class(es) with probability 1. For further details see section 6.7.

The option ‘posterior=(varlist)’ is needed for step-three modeling. Rather than defining the number of categories of the ordinal/nominal variable concerned, one specifies the variables in the input data file containing the posterior class membership probabilities for the K classes; that is, the variables obtained by writing the classification information from the step-I analysis to an output data file. This option can be used for any of the ordinal/nominal latent variables in the model.

Example:

\[
\text{latent gclass group 3 coding=last, cfactor continuous, dfactor ordinal 3 score=(0, 1, 3)};
\]

4.2.6 Survey options

- \text{stratumID <varname>};
- \text{psuid <varname>};
- \text{samplingweight <varname> <rescale>};
- \text{populationsize <varname>};
- \text{rsweights <varlist>};

These five keywords are used to specify the sampling design for complex sampling survey data sets or other types of data with dependent observations. ‘Samplingweight’ provides the name of the variable to be used as a sampling weight and indicates whether it should be rescaled. Replicate sampling weights available for variance estimation can be defined with the rsweights keyword.

4.2.7 Options for Markov models

Two options which are specific for Markov models are

- \text{timeinterval <varname>}
- \text{timelabel <varname>}

Specification of the variable indicating the length of the time-intervals between measurement occasions is used for defining continuous-time latent Markov models. In fact, using this option is also used to indicate that the specified model is a continuous-time Markov model.
With ‘timelabel’ one can select a variable from the input data file that should be used to label the discrete-time categories in the various output sections (such as in Profile-longitudinal).

4.2.8 Other options

Five other options are

- select <varname [=] 99 [to] 99 ... 99 [to] 99>
- holdout <cases|replications> <varname [=] 99 [to] 99 ... 99 [to] 99>
- validationid <varname>
- simulationid <varname>
- imputationid <varname>

The command ‘select’ allows particular records to be selected for analysis based on the values / categories of a specified selection variable. The selection variable may be either a numeric or a string variable. Records having any of the values specified are selected for analysis. Values need to be separated by one or more spaces. For character variables, valid character strings need to be placed in single quotes and separated by spaces (REGION = ‘North’ ‘South’). For a numeric selection variable a range of values can be specified using the keyword ‘to’. The ‘=’ may be omitted.

Some examples:

```
select AGE = 18 19 20 21 22 23 24;
select AGE = 18 to 24;
select REGION = ‘North’ ‘South’;
select AGE 18 to 24 55 to 64 . ;
```

The last example illustrates the use of ‘.’ to indicate the inclusion of cases containing a missing value on the selection variable AGE.

The command ‘holdout’ can be used to specify which cases or replications should not be used for parameter estimation. Separate statistics are provided for the holdout cases/replications. For holdout replications one obtains prediction statistics, and for holdout cases one obtains chi-squared, log-likelihood, classification, and prediction statistics. The specification is the same as for the ‘select’ command.

The command ‘validationid’ can be used to specify an id variable defining the user-specified folds for n-fold validation log-likelihood computation. This serves as an alternative to the ‘validationLL=%d’ output option, which yields a random split of the sample over the folds.

The command ‘simulationid’ is used to indicate that the specified data file contains a number of stacked data file which are distinguished by the specified identifier. The program will analyze
each data set separately and combine the results. Parameter estimates (including Profile, ProbMeans, and EstimatedValues) and statistics are averages across data sets, and standard errors are standard deviation across data sets.

The command ‘imputationid’ is used to analyze multiply imputed data sets. The program will analyze each complete data set separately and combine the results using the well-known multiple imputation formulae.

4.3 Equations section

The ‘equations’ section specifies the model to be estimated using a set of equations, a set of restrictions on parameters (if any), and a set of starting values for parameters (optional). Three types of equations are used:

- regression equations, defining the generalized linear model for each latent and dependent variable,
- variance ‘equations’ for continuous latent variables and dependent variables which are continuous or overdispersed counts,
- covariance and association ‘equations’ for latent variables and for dependent variables of the same scale types (if included), and
- log-linear scale factor equations.

The equations utilize the names of the latent, dependent, and independent variables defined in the ‘variables’ section. In addition, the reserved name ‘1’ denotes the vector of ones corresponding to the intercept of a regression equation. A special feature of LG-Equations is that regression parameters, (co)variances, and associations can be allowed to vary across levels of certain other variables, for example, across latent classes or across categories of a nominal independent variable serving as grouping variable. This is what we will refer to as ‘conditional effects’.

Labels may be assigned to the model parameters. As is shown below, such labels can be used to define fixed-value, equality, ratio, and monotonicity restrictions immediately following the model equations, as well as to specify starting values for particular model parameters. Certain restriction may also be imposed by the parameter labels themselves. For example, using the label ‘(1)’ means that the parameter(s) concerned should be fixed to 1, using the same label ‘(a)’ for two different (sets of) parameters equates the two, using the label ‘(2 a)’ indicates that parameter concerned is twice as large another parameter labeled ‘(a)’, and using the label ‘(+’) means that a monotonicity or nonnegativity restriction should be imposed.

4.3.1 Regression equations

Regression equations have the following form:

<varname> <- <(label)> <varlist> < | varlist> + … +… ;
It is the symbol ‘<-’ that distinguishes regression equations from the variance and covariance/association equations. The left-most variable in a regression equation (i.e., the variable positioned at the left-hand side of the ‘<-’ symbol) contains the name of a latent or a dependent variable. To the right of the ‘<-’ symbol, is the linear predictor for the regression model with terms separated by a ‘+’. Each term contains one or more variables – a single variable in the case of a main effect and multiple variables for an interaction term. As mentioned above, the internal variable name ‘1’ is used to denote the vector of ones corresponding to the intercept. The second optional part of a term ‘| varlist’ makes it possible to indicate that parameter values vary across the categories of another (set of) variable(s); that is, to define ‘conditional effects’.

Example:

response <- 1 + (a) class + gender + class gender + (d) age | class;

The regression model for the dependent variable ‘response’ contains an intercept (‘1’), main effects of ‘class’ and ‘gender’, a class-gender interaction (‘class gender’ or ‘class * gender’), and an ‘age’ effect that differs across the categories of the variable ‘class’ (‘age | class’). The labels ‘a’ and ‘d’ could be used to specify restrictions or starting values (see below).

4.3.2 Variances

The variance equations for continuous latent variables and for dependent variables which are continuous or overdispersed counts have the following form:

<(<label>) <varname> < | varlist>;

where ‘| varlist’ can be used to condition the value of variance on the value of another (set of) variable(s).

Example:

(s) response | class gender;

This defines a heteroskedastic error variance for the continuous dependent variables ‘response’, which varies across ‘class’ and ‘gender’.

4.3.3 Covariances and associations

Covariances and associations between latent variables and between dependent variables of the same scale type can be included using terms like:

<(<label>) <varname> <-> <varname> <| varlist>;

It is the symbol ‘<->’ that distinguishes covariance/association equations from regression and variance equations.
Example:

\[ \text{response1} \leftrightarrow \text{response2} | \text{gender}; \]

Assuming that ‘response1’ and ‘response2’ are both continuous dependent variables, this defines a free covariance between these variables which depends upon the variable gender.

Note that for categorical latent/dependent variables the “<-” option yields a two-variable log-linear association term. It is also possible to include higher-order log-linear associations. This is achieved by separating the variables by “<-”. An example of a three-variable association term is

\[ \text{catresp1} \leftrightarrow \text{catresp2} \leftrightarrow \text{catresp3}; \]

4.3.4 Scale factor equations

For nominal, ordinal, and the choice-type (choice, ranking, and rating) dependent variables, one can specify log-linear scale factor models, which also appear in the LG-Syntax equations section. A scale factor equation is identifiable by the symbol “<<” between the dependent variable at the left-hand side and the linear term with the relevant latent and independent variables at the right-hand side; that is,

\[ <\text{varname}> \leftrightarrow(<\text{(label)}><\text{varlist}><|\text{varlist}> + ... +... ; \]

An example of a scale factor equation is:

\[ \text{response} \leftrightarrow \text{Class} + \text{gender}; \]

Note that typically the intercept term will be excluded from the scale factor model for identification purposes. All options available for regression equations can also be used with scale factor equations, such as conditional effects, interactions, and parameter constraints, except for most of the “~” parameter options.

4.3.5 Using lists of variables

To ease the model specification process, LG-Equations allows you to define models containing multiple equations in a compact way. Specifically, you can provide a list of the left-most variables, where the symbol ’-‘ can be used to denote a range of such (dependent) variables in the list. For example, consider the following four equations:

\[ \text{response1} \leftarrow 1 + \text{class}; \]
\[ \text{response2} \leftarrow 1 + \text{class}; \]
\[ \text{response3} \leftarrow 1 + \text{class}; \]
\[ \text{response4} \leftarrow 1 + \text{class}; \]

These equations can be re-expressed using a list in a single equation as:

\[ \text{response1 response2 response3 response4} \leftarrow 1 + \text{class}; \]
or reduced even further using the ‘-‘ option:

\[
\text{response1 - response4} \leftarrow 1 + \text{class};
\]

The last specification is equivalent to the previous ones provided that the dependent variables response1, response2, response3, and response4 are ordered in this way in the ‘dependent’ list in the ‘variables’ section.

4.3.6 Restrictions

The four types of restrictions that can be imposed on labeled parameters are:

- Equality: \(<\text{label}>\{\%d,\%d\} = <\text{label}>\{\%d,\%d\} >
- Fixed value: \(<\text{label}>\{\%d,\%d\} = \%f | \{\ldots\} | <\text{filename}> >
- Ratio: \(<\text{label}>\{\%d,\%d\} = \%f <\text{label}>\{\%d,\%d\} >
- Monotonicity: \(<\text{label}>\{\%d,\%d\} = < - / + / + - / + >

Note that a ‘label’ will often be used to refer to a set of parameters. The appended ‘[\%d,\%d ]’ represents a parameter(s) index that can be used to access a single or subset of parameters within a set. The first index number refers to the (joint) category number of the conditioning variable(s) (the variables appearing after the ‘|’ symbol); the second index refers to the parameter number within a set (before conditioning).

You can view the index associated with each parameter displayed in the ‘Parameters Output’. The hidden column “Index” can be made visible by selecting “Index” from the popup menu obtained by right clicking in the Parameters output.

To illustrate how restrictions are imposed, suppose we have the following simple latent class model

\[
\text{class} \leftarrow 1;
\text{y1} \leftarrow (a) 1 | \text{class};
\text{y2} \leftarrow (b) 1 | \text{class};
\text{y3} \leftarrow (c) 1 | \text{class};
\]

where ‘class’ is a 3-class latent variable, and ‘y1’, ‘y2’, and ‘y3’ are three trichotomous nominal dependent variables. The last category of ‘y1’, ‘y2’, and ‘y3’ is used as the reference category (‘parameters=last’ or ‘coding=last’). Here are some example restrictions illustrating the parameter indexing. With

\[
a = b;
\]

the parameters in set labeled ‘a’ are equated to the parameters in set ‘b’, which in this model make the response probabilities equal for ‘y1’ and ‘y2’. Note that this constraint can alternatively
be imposed without this equation by simply changing the label ‘b’ to ‘a’ so that the label ‘a’ is re-used. The restriction

\[ a[3,1] = a[2,1]; \]

equates the first parameter in set ‘a’ for subgroups 2 and 3; that is, the first intercept coefficient (the logit of responding in the first instead of the last category of ‘y1’) is equated for classes 2 and 3. With

\[ a[3,] = a[2,]; \]

all parameters in set ‘a’ are made equal for subgroups 2 and 3. This restriction equates all intercept coefficients for classes 2 and 3, which implies that the response probabilities for ‘y1’ are equal for these two classes. This constraint can be stated more simply as ‘a[3] = a[2];’.

An example of a fixed value constraint is

\[ a[,1] = 1; \]

which fixes the first parameter in the set to 1 for all subgroups (here for all classes). An example of a ratio constraint is

\[ a[1] = 2 a[2]; \]

implying that the logit intercept parameters for class 1 are twice as large as the ones for class 2.

If all parameters in a set should take on a fixed value, it is also possible to define the fixed values simultaneously. Suppose the parameter set labeled ‘a’ contains 3 parameters. We can then define the fixed value constraints on the parameters in this set as follows:

\[ a =\{-2 0 2\}; \]

that is, by putting them between braces ‘{…}’. Alternatively, one may read the fixed values from a file; that is,

\[ a =’filewithconstraints.txt’; \]

Constraints are evaluated in the order in which they are specified. Once a parameter has been included to the right of the ‘=’, generally speaking, it should not be used on the left-hand side subsequently, as doing so may serve to alter the first constraint in ways that are not desired. For example, suppose it is desired to set 3 parameters, labeled ‘a’, ‘b’, and ‘c’ equal to each other. This can be done correctly as follows:

\[ b = a; c = a; \]
(Alternatively, the labels ‘a’, ‘b’ and ‘c’ could all be replaced by a common label, say, ‘a’.) However, the following would be incorrect as it would not achieve the desired result:

\[ a = b; \quad b = c; \]

since ‘a’ was set to ‘b’ prior to ‘b’ being changed to equal ‘c’, ‘a’ and ‘b’ will not end up being equal. Reversing the two restrictions

\[ b = c; \quad a = b; \]

yields the correct result since ‘a’ is set to ‘b’ which is equal to ‘c’.

The fact that constraints are evaluated in the order in which they are specified may be used to reduce the number of constraints that would otherwise need to be specified. For example, the result of

\[ a = b; \quad a[1] = 0; \]

is that all ‘a’ terms are equated to the corresponding ones in ‘b’, except for the first subgroup (first class in our example model) which is equated to the value 0.

The working of equality, fixed value, and ratio constraints should be quite clear from the above explanation and examples. Slightly more complex are the monotonicity constraints ‘+’, ‘-’, ‘+-’ and ‘-+’, the exact meaning of which depends on the scale types of the variables involved and whether they are specified for a single parameter or for a parameter set. Let us illustrate this point for the constraint ‘+’, which yields monotonically nondecreasing parameters (they either increase or remain equal across categories) when the constraint is used for a nominal variable, and which yields nonnegative parameters (positive or zero) otherwise. As an example, assume we have the following regression model:

\[ \text{response} < - 1 \mid \text{gender} + (b) \text{time} \mid \text{gender}; \]

where ‘time’ is a nominal independent variable. The constraint

\[ b = +; \]

yields a monotonically nondecreasing time effect for both males and females. The constraint

\[ b[1] = +; \]

yields a monotonically nondecreasing time effect for the first category of gender, say for males if these have the score 1 and females the score 2. However, the following constraint for a single parameter

\[ b[1,1] = +; \]
implies that the first time parameter for males should be nonnegative, the meaning of which depends on the coding used for the time variable. For ‘coding=last’ it means that the first category is at least as large as the last category. Note that if time is numeric instead a nominal independent variable (and response is not nominal), the constraints ‘b[1] = +’ and ‘b[1,1] = +;’ would be identical.

The function of the constraint ‘-’ is the same as ‘+’ except for the fact that the direction of the ordering or sign is reversed. The constraints ‘+–’ and ‘–+’ are single peakness restrictions, which are relevant for nominal variables only. Single peakness means that the parameters are monotonic nondecreasing until a certain category and from that point they are monotonic nonincreasing (‘+–’), or the other way around when ‘–+’ is used. In other words, for ‘+–’, once there is a decrease, there can be no further increase. The active-set estimation algorithm attempts to find the optimal point at which the monotonicity pattern changes.

4.3.7 Starting values

Starting values for labeled parameters are provided in a way that is identical to a fixed value restriction, except that ‘–=’ is used instead of ‘=’:

\[
<\text{label}>[\%d,\%d] \; \sim \; <\%f> \; | \{\ldots\} \; | \text{filename};
\]

Alternatively, parameter values may be provided for all free parameters simultaneously by placing the values within parentheses at the bottom of the syntax. This format is generated automatically when the ‘Save Model with Parameters’ option is used. For further details see section 7.1.

4.3.8 Features for Markov models

In a latent Markov model there are two separate regression equations for the dynamic latent variable: one for the initial state at time point 0 and one for the state at time point t conditional on the state at time point t-1. This requires that three ‘versions’ of the same dynamic latent variable be used in the Markov model equations. These are distinguished by appending [=0] and [-1] to the latent variable name when referring to the initial state (T=0) and the previous time point (T=t-1), respectively. Nothing is appended when referring to the state at T=t.

An example of a simple latent Markov model for a single response variable is:

```
variables
  dependent Y nominal;
  latent Class dynamic 3;
equations
  Class[=0] <- 1;
  Class <- 1 | Class[-1];
  Y <- 1 | Class;
```

The first equation is the logit equation for the initial state probabilities at T=0, the second is the logit equation for the transition probabilities between T=t-1 and T=t, and the third equation links the state at time T=t to the dependent variable at time point T=t.
4.3.9 Special parameter coding using the tilde character (~)

LG-Syntax implements various special types of coding schemes that may be requested by appending a term starting with a ‘~’ to the parameter label. The complete list of special coding options is:

- ~transition or ~tra
- ~error or ~err
- ~difference or ~diff
- ~nominal or ~nom
- ~ordinal or ~ord
- ~squared or ~sqr
- ~ful or ~full
- ~wei or ~weight
- ~int or ~intercept
- ~mis or ~missingvaluedummy

The option ‘~tra’ can be used for modeling transition probabilities in latent Markov models and should be combined with a conditioning of the parameter set concerned on the state at T=t-1. This will yield dummy coding with a running reference category that changes with the value of the latent state at the previous occasion. The running reference category is the ‘no change’ category; that is, the diagonal element of the associated squared table. An example is

Class <- (~tra) 1 | Class[-1] + (~tra) time | Class[-1] + (~tra) gender | Class[-1];

The estimated intercept, time, and gender coefficients can be interpreted as effects on the logit of experiencing a transition of a certain type rather than experiencing no transition. It should be noted that in continuous-time Markov models the use of ~tra and the conditioning on the state at the previous occasion is obligatory.

The term ‘~err’ works in the same way as ‘~tra’, with the difference that it is used for the relationship between a nominal latent and a nominal dependent variable with the same numbers of categories. The running reference category is the ‘no error’ category, which is the diagonal element of the associated squared table.

Example:

Y <- (~err) 1 | Class;

The term ‘~dif’ requests a coding of parameters for nominal variables in terms of differences between adjacent categories, rather than using effects or dummy coding. This could thus also be called adjacent-category coding.

The term ‘~nom’ can be used to include a nominal term in a model for an ordinal dependent/latent variable. In other words, even if the left-hand side variable in the equation
concerned has been defined to be ordinal in the ‘variables’ section, it will be treated as nominal in a parameter set containing the ‘~nom’ option.

Similarly, the term ‘~ord’ can be used to include an ordinal term in a model for a nominal dependent/latent variable; that is, even if the left-hand side variable has been defined to be nominal, in the parameter set concerned it will be treated as (fixed-scored) ordinal.

The term ‘~sqr’ can be used to include a term with squared scores in a model for an ordinal dependent or latent variable. In such a term squared category scores will be used for the left-hand side variable.

The option ‘~ful’ yields a full set of unrestricted parameters for a nominal dependent or latent variable; that is, a parameter set without identifying effects or dummy coding constraints. The user has complete control over coefficients and can fix and equate any of the parameters.

The option ~wei is used to define a vector with cell weights. These are fixed multiplicative which can be used in models for latent and dependent variables which are nominal or ordinal. A cell weight is equivalent to what is known as an offset in generalized linear modeling, but then in exponential form; thus, a cell weight equals exp(offset) or an offset equals log(cell weight). Among other uses, cell weights can be used to fix probabilities to a specific value.

The option ~int is used to indicate that the term concerned should be interpreted as being the intercept. This is needed in continuous-time Markov models without an intercept term in the model for the transition intensities.

The option ~mis will create a dummy variable for the missing value category of the independent variable concerned. It takes on the value 1 if the independent variable is missing and 0 otherwise. Note that missing values on the independent variable are treated in the usual way (mean imputation for numeric and average effect for nominal).

4.3.10 Starting values and saved parameters between ‘{ }’ or in separate text file

The Save Syntax with Parameters option in the File Save Syntax menu command allows saving estimated Parameters in addition to the model settings and equations. An example of a syntax model which was saved using this option is:

```
equations
  response <- 1 | Class + year + religion ;
{
  2.13422872761534
  0.05006324201778482
  0.111864374525874
  -0.3854812362408223
  0.009992798082358045
  -0.3465260120313344
  0.2369151273770218
  -0.2970236345163084
```
The numbers between the braces (curly brackets) ‘{...}’ are the saved parameter values, which serve as starting values when the restored model is rerun. Note that it is also possible to specify starting values in a text file, that is,

```
equations
  response <- 1 | Class + year + religion;
  'filewithstartingvalues.txt'
```

From the above input it is not immediately clear which number corresponds to which model parameter. This information may be obtained in the Parameters output, using the popup menu to request that the hidden column ‘Internal’ appear in the output. Upon selecting ‘Internal’, the parameter column containing sequential numbering from 1 to the number of free parameters will appear in the Parameters output which corresponds to the order in which the parameters should appear between the braces.

The inclusion of parameter values between the braces can be used in various applications. In particular, it can be used to:

- restore model results without the necessity to re-estimate the model from scratch (rerun model, possibly with number of iterations set to 0),
- generate additional output from a model that has already been estimated (rerun model with additional output and outfile options),
- restart the estimation process from the current termination point if a model did not converge (restart model),
- score one or more alternative data files using an estimated model (link a save model to other data, which should be done by editing the lgs file with an external editor to set the number of iterations to 0, and rerun),
- provide starting values for all parameters without the need to define parameter labels (provide the user-defined starting values between ‘{ }’),
- define the population from which data should be simulated in MC simulation studies and power computations (provide the population values between ‘{ }’).

In most cases, the internal parameters between the braces will match the parameters reported in the Parameters output. There is, however, one exception, which occurs when monotonicity constraints are imposed. In that case, because of an internal reparameterization of the model, the internal parameters will be differences between categories rather than the reported parameters themselves. Moreover, when numerical integration is used, the (co)variances of the latent variables are reparameterized using a Cholesky decomposition. The internal parameters are then not (co)variances but elements of the Cholesky decomposed (co)variance matrices.

It should be noted that these two reparameterizations should not only be taken into account when defining the starting values, but also when imposing parameter constraints.
4.3.11 Unallowed model specifications

Certain model specifications are not allowed because they yield meaningless models, or models that are not yet implemented in Latent GOLD. When such model specification statements are encountered, the program will report an error message indicating the line numbers containing the equation in which the error occurs. In particular, it is meaningless and thus not allowed:

1. to use lower-level latent variables as right-hand-side (RHS) variables in equations for higher-level latent variables
2. to use continuous latent variables or numeric independent variables as conditioning variables (after a ‘|’)
3. to define variance equations for latent variables that are not continuous or for dependent variables that are not continuous or overdispersed counts

Some specifications are not allowed because the associated models are not currently implemented in Latent GOLD 5.0. For example, it is not allowed:

1. to define covariances or associations between pairs of dependent variables which are not both (non-censored and non-truncated) continuous or both categorical (nominal or ordinal)
2. to define covariances or associations between pairs of latent variables of the same level which are not both continuous or both categorical (nominal or ordinal)
3. to use a dependent variable as a RHS variable in regression equations for another dependent variable or for a latent variable
4. to use a continuous latent variable as a RHS variable in the regression model for another continuous latent variable of the same level

In most cases, the latter restrictions can be circumvented by means of tricks. Restrictions 1 and 2 are similar in the sense that they concern (non-directed) associations between variables of different scale types. One way to define associations in such situations is to include an additional latent variable in the model which affects the two variables that you desire to be associated. Another way is to include a direct effect of one dependent (latent) variable on the other. While this is not allowed for dependent variables because of restriction 3, this restriction can also be circumvented. More specifically, specification 3 can be accomplished by creating a second copy in the data file of the dependent variable concerned, and use the copy as an independent variable in the model. An alternative approach is to define a latent variable which is perfectly related to the dependent variable concerned, and use this ‘quasi-latent’ variable as a predictor in the equation for the other dependent or the latent variable. The fourth restriction can be circumvented by defining the explanatory latent variable as a higher-level latent variable.

4.4 Selecting a keyword, variable, file, or model

As indicated above, Latent GOLD 5.0 utilizes various reserved words. A list of all of these keywords may be viewed with a right-click in the Syntax Window and selecting ‘Keywords’. This will cause a list of all the reserved words to appear in a pop-up menu. Any of these may be
selected. Upon selection it will be placed in the LG-Syntax window according to the position of the cursor when the mouse was initially right-clicked. This feature can be used to insert a keyword in the appropriate place or to replace highlighted text with the selected keyword.

The same menu also allows selecting ‘Variables’. This causes a list of all variables in the data file to appear in a popup menu. Any of the may be selected and this placed in the LG-Syntax window. Two similar options are the possibility to select a ‘File’ (for the outfile option) and the possibly to select a ‘Model’ (for the montecarlo options in which one has to refer to another model).
5 Model Examples

The best way to learn the syntax is by means of examples. Examples of the most important applications can be found in Help Examples menu entries for both GUI and syntax models. When selecting a syntax example a saved ‘lgs’ file will be opened. Because these examples are attached to data sets, these can be run, possibly after modifying parts of the model specification.

Some examples of ‘equations’ sections are given below which illustrate various aspects of the LG-Syntax module. This list of example models is not meant to be comprehensive. Many more model types can be defined with the Syntax module than are illustrated below.

5.1 Cluster models

We will illustrate some Cluster models similar to models that can be defined with the Cluster module, as well as some more extended models containing constraints, interactions, and conditional effects.

5.1.1 Unrestricted cluster model

An unrestricted cluster model is defined as follows:

```
Cluster <- 1;
Item1 <- 1 + Cluster;
Item2 <- 1 + Cluster;
Item3 <- 1 + Cluster;
Item4 <- 1 + Cluster;
```

or more compactly as

```
Cluster <- 1;
Item1 - Item4 <- 1 + Cluster;
```

For the nominal latent variable called ‘Cluster’ we have a (logistic) regression model containing only an intercept; for the four indicators we have (logistic) regression models containing an intercept (indicated with the ‘1’) and an effect of the latent variable ‘Cluster’.

An equivalent formulation – for all scale types except ordinal indicators – is:

```
Cluster <- 1;
Item1 <- 1 | Cluster;
Item2 <- 1 | Cluster;
Item3 <- 1 | Cluster;
Item4 <- 1 | Cluster;
```

Here, we illustrate the possibility that an effect differs across categories of another variable. More specifically, we indicate that the intercepts for the items in the regression models differ
across clusters. This is similar to the conditional-effects parameterization used in the Latent GOLD Regression module (i.e., class dependent intercepts).

A set of variance equations are required for continuous indicators or for overdispersed counts, which have the form:

\[
\text{Item1 | Cluster;}
\text{Item2 | Cluster;}
\text{Item3 | Cluster;}
\text{Item4 | Cluster;}
\]

where the within-cluster variances are assumed to vary across clusters. The ‘| Cluster’ clause is omitted when variances are assumed to be class independent. Note that the complete model for continuous or overdispersed count variables can be specified very compactly as

\[
\text{Cluster <- 1;}
\text{Item1 - Item4 <- 1 | Cluster;}
\text{Item1 - Item4 | Cluster;}
\]

5.1.2 Restricted cluster model

Fixed-value and equality constraints on probabilities are most easily defined using the ‘conditional effects’ specification for the logit parameters in the model for the item responses. For example, a model with equal response probabilities for the first two and last two items is obtained with the following specification:

\[
\text{Cluster <- 1;}
\text{Item1 <- (a) 1 | Cluster;}
\text{Item2 <- (a) 1 | Cluster;}
\text{Item3 <- (b) 1 | Cluster;}
\text{Item4 <- (b) 1 | Cluster;}
\]

The logit parameters are restricted to be equal by using the same label for the effects concerned. You may also formulate the equality restrictions more explicitly:

\[
\text{Cluster <- 1;}
\text{Item1 <- (a) 1 | Cluster;}
\text{Item2 <- (b) 1 | Cluster;}
\text{Item3 <- (c) 1 | Cluster;}
\text{Item4 <- (d) 1 | Cluster;}
\text{b = a;}
\text{d = c;}
\]

Suppose you wish to equate these probabilities only for cluster 1. This is achieved with the restrictions:

\[
b[1,] = a[1,];
\text{d[1,] = c[1,];}
\]
A model with fixed zeroes for certain response probabilities can be obtained by restricting the corresponding (logit) parameters to –100 or 100. For example:

\[
a[1,] = 100;
d[2,] = -100;
\]

Assuming that you have dichotomous indicators and use effects or dummy-last coding, this yields fixed zero probabilities for the last category for item 1 for cluster 1 and for the first category of item 4 for cluster 2.

### 5.1.3 Cluster model with local dependencies

A local dependency between two categorical indicators or a covariance between two continuous indicators can be included using a term like:

\[
\text{Item1} \leftarrow \text{Item2};
\]

A class-specific direct effect is defined as follows:

\[
\text{Item1} \leftarrow \text{Item2} | \text{Cluster};
\]

Suppose that the local dependency should only be present in cluster 2 in a 2-cluster model. This can be achieved as follows:

(a) \[
\text{Item1} \leftarrow \text{Item2} | \text{Cluster};
a[1,] = 0;
\]

That is, by indicating that the local dependency differs across clusters and then restricting the effect to equal 0 in cluster 1.

### 5.1.4 Cluster model with covariates

Inclusion of an active covariate involves a change only in the regression equation for the clusters. An example is:

\[
\text{Cluster} \leftarrow 1 + \text{sex} + \text{age};
\]

This applies regardless of whether sex and age are defined to be nominal or numeric independent variables. The interaction between sex and age can also be included easily:

\[
\text{Cluster} \leftarrow 1 + \text{sex} + \text{age} + \text{sex age};
\]

or equivalently as

\[
\text{Cluster} \leftarrow 1 + \text{sex} + \text{age} + \text{sex} \times \text{age};
\]
A direct effect of a covariate on an indicator is specified as follows

\[
\text{Item3} \leftarrow 1 + \text{Cluster} + \text{sex};
\]

and an effect that varies across clusters as

\[
\text{Item3} \leftarrow 1 + \text{Cluster} + \text{sex} \mid \text{Cluster};
\]

5.1.5 **Multiple-group cluster model**

A multiple-group LC model is a model in which certain parameters differ across observed subgroups, as defined by a grouping variable. Such a model may be specified easily with the syntax by indicating that the grouping variable is a nominal independent variable and by making use of the conditional-effects option. An example of an *unrestricted* multiple-group model where the subgroups are defined by the variable ‘sex’ is:

\[
\begin{align*}
& \text{Cluster} \leftarrow 1 \mid \text{sex}; \\
& \text{Item1} \leftarrow 1 \mid \text{sex} + \text{Cluster} \mid \text{sex}; \\
& \text{Item2} \leftarrow 1 \mid \text{sex} + \text{Cluster} \mid \text{sex}; \\
& \text{Item3} \leftarrow 1 \mid \text{sex} + \text{Cluster} \mid \text{sex}; \\
& \text{Item4} \leftarrow 1 \mid \text{sex} + \text{Cluster} \mid \text{sex};
\end{align*}
\]

or in a slightly different parameterization (which is equivalent to the above except in the case of ordinal indicators)

\[
\begin{align*}
& \text{Cluster} \leftarrow 1 \mid \text{sex}; \\
& \text{Item1} \leftarrow 1 \mid \text{Cluster} \mid \text{sex}; \\
& \text{Item2} \leftarrow 1 \mid \text{Cluster} \mid \text{sex}; \\
& \text{Item3} \leftarrow 1 \mid \text{Cluster} \mid \text{sex}; \\
& \text{Item4} \leftarrow 1 \mid \text{Cluster} \mid \text{sex};
\end{align*}
\]

It should be noted that (except for ordinal indicators) these models are also equivalent to:

\[
\begin{align*}
& \text{Cluster} \leftarrow 1 + \text{sex}; \\
& \text{Item1} \leftarrow 1 + \text{Cluster} + \text{sex} + \text{Cluster} \mid \text{sex}; \\
& \text{Item2} \leftarrow 1 + \text{Cluster} + \text{sex} + \text{Cluster} \mid \text{sex}; \\
& \text{Item3} \leftarrow 1 + \text{Cluster} + \text{sex} + \text{Cluster} \mid \text{sex}; \\
& \text{Item4} \leftarrow 1 + \text{Cluster} + \text{sex} + \text{Cluster} \mid \text{sex};
\end{align*}
\]

That is, sex is used as a covariate having direct effects on all items, as well as interactions with the latent variable ‘Cluster’. This shows that conditional effects and interactions may yield equivalent models which are, however, parameterized in a different way.
5.2 Discrete-factor models and other models with multiple discrete latent variables

In this section we describe some models with discrete latent variables that are ordinal; that is, models similar to those that can be defined with the DFactor module.

Note that whereas in the GUI DFactor module the latent variables named DFactor1, DFactor2, etc. are assumed to be dichotomous or ordinal latent variables, in the syntax these (or other names) can be assigned to nominal latent variables, which makes a difference when they have more than 2 levels.

5.2.1 Single DFactor model

A DFactor model with a single ordinal latent variable is obtained with equations similar to the cluster models discussed above. The linear term for the indicators will typically be ‘1 + DFactor’,

\[
\begin{align*}
\text{DFactor} & \leftarrow 1; \\
\text{Item1} & \leftarrow 1 + \text{DFactor}; \\
\text{Item2} & \leftarrow 1 + \text{DFactor}; \\
\text{Item3} & \leftarrow 1 + \text{DFactor}; \\
\text{Item4} & \leftarrow 1 + \text{DFactor}; \\
\text{Item5} & \leftarrow 1 + \text{DFactor};
\end{align*}
\]

Although it is allowed to use the conditional-effects specification ‘1 | DFactor’ as we did in some of the cluster models presented above, with an ordinal latent variable this yields a different model – one where the ordinal latent variable is changed into nominal for the associated effects.

5.2.2 Exploratory two-DFactor model

A standard 2-DFactor model obtained with the GUI DFactor model can be defined with the following set of equations:

\[
\begin{align*}
\text{DFactor1} & \leftarrow 1; \\
\text{DFactor2} & \leftarrow 1; \\
\text{Item1} & \leftarrow 1 + \text{DFactor1} + \text{DFactor2}; \\
\text{Item2} & \leftarrow 1 + \text{DFactor1} + \text{DFactor2}; \\
\text{Item3} & \leftarrow 1 + \text{DFactor1} + \text{DFactor2}; \\
\text{Item4} & \leftarrow 1 + \text{DFactor1} + \text{DFactor2}; \\
\text{Item5} & \leftarrow 1 + \text{DFactor1} + \text{DFactor2};
\end{align*}
\]

or more compactly (but perhaps less readable) as

\[
\begin{align*}
\text{DFactor1} - \text{DFactor2} & \leftarrow 1; \\
\text{Item1} - \text{Item5} & \leftarrow 1 + \text{DFactor1} + \text{DFactor2};
\end{align*}
\]
5.2.3 Confirmatory two-DFactor model

A confirmatory 2-DFactor model with items 1 and 2 loading on DFactor 1, items 3 and 4 on DFactor 2, item 5 on both, and an association between these 2 DFactors is obtained as follows:

\[
\begin{align*}
&\text{DFactor1} \leftarrow 1; \\
&\text{DFactor2} \leftarrow 1; \\
&\text{DFactor1} \leftarrow \text{DFactor2}; \\
&\text{Item1} \leftarrow 1 + \text{DFactor1}; \\
&\text{Item2} \leftarrow 1 + \text{DFactor1}; \\
&\text{Item3} \leftarrow 1 + \text{DFactor2}; \\
&\text{Item4} \leftarrow 1 + \text{DFactor2}; \\
&\text{Item5} \leftarrow 1 + \text{DFactor1} + \text{DFactor2};
\end{align*}
\]

5.2.4 Confirmatory two-DFactor with a path model for the DFactors

A slight modification of the above confirmatory 2-DFactor model is obtained by replacing the association between the DFactor1 and DFactor2 by an effect of DFactor 1 on DFactor2. Such a path model is defined as follows:

\[
\begin{align*}
&\text{DFactor1} \leftarrow 1; \\
&\text{DFactor2} \leftarrow 1 + \text{DFactor1}; \\
&\text{Item1} \leftarrow 1 + \text{DFactor1}; \\
&\text{Item2} \leftarrow 1 + \text{DFactor1}; \\
&\text{Item3} \leftarrow 1 + \text{DFactor2}; \\
&\text{Item4} \leftarrow 1 + \text{DFactor2}; \\
&\text{Item5} \leftarrow 1 + \text{DFactor1};
\end{align*}
\]

5.2.5 Confirmatory two-DFactor model with a path model containing independent variables

An example of a similar path model, but now with three independent variables (sex, age, and education) is defined as follows:

\[
\begin{align*}
&\text{DFactor1} \leftarrow 1 + \text{sex} + \text{educ} + \text{age} + \text{sex educ}; \\
&\text{DFactor2} \leftarrow 1 + \text{DFactor1} + \text{educ} + \text{age}; \\
&\text{Item1} \leftarrow 1 + \text{DFactor1}; \\
&\text{Item2} \leftarrow 1 + \text{DFactor1}; \\
&\text{Item3} \leftarrow 1 + \text{DFactor2}; \\
&\text{Item4} \leftarrow 1 + \text{DFactor2}; \\
&\text{Item5} \leftarrow 1 + \text{DFactor1};
\end{align*}
\]

As can be seen DFactor1 is regressed on sex, educ, age, and the sex-educ interaction term, and DFactor1 on DFactor2, educ and age.
5.3 Factor analysis and IRT models

Now we will provide examples of models with continuous latent variables. These include models that can be estimated with the GUI models, but also more extended models.

5.3.1 Single factor or IRT model

A simple one-factor or one-dimensional IRT model is defined as follows:

(1) Factor;
  Item1 <- 1 + Factor;
  Item2 <- 1 + Factor;
  Item3 <- 1 + Factor;
  Item4 <- 1 + Factor;

where the first statement restricts to 1 the variance for Factor for purposes of identification. The alternative convention for identifying this model, where the continuous factor variance is freely estimated and one loading is fixed to 1 is specified as follows:

Factor;
  Item1 <- 1 + (1) Factor;
  Item2 <- 1 + Factor;
  Item3 <- 1 + Factor;
  Item4 <- 1 + Factor;

A finite mixture variant of this model in which only the factor means differ across latent classes could be obtained by including a nominal latent variable (say Cluster) and adding the equations:

Cluster <- 1;
Factor <- Cluster;

Note that the intercept in the regression model for the continuous factor is omitted (set to 0) in order to identify the factor means.

5.3.2 Confirmatory 2-factor or 2-dimensional IRT model

A confirmatory 2-factor model for 6 indicators may have the following form:

Factor1;
Factor2;
Factor1 <-> Factor2;
Item1 <- 1 + (1) Factor1;
Item2 <- 1 + Factor1;
Item3 <- 1 + Factor1;
Item4 <- 1 + (1) Factor2;
Item5 <- 1 + Factor2;
Item6 <- 1 + Factor2;

A *mixture variant* with class-specific factor means would include these three additional equations:

Cluster <- 1;
Factor1 <- Cluster;
Factor2 <- Cluster;

5.3.3 **Factor analysis or IRT models with covariates**

Covariate effects on a continuous factor are included in the model as:

Factor <- sex + age;

and covariate effects on an indicator as:

Item3 <- 1 + Factor + sex;

In a mixture variant with also a Cluster effect on the factor means, the regression model for the continuous factor would have the following form:

Factor <- Cluster + sex + age;

5.3.4 **Latent growth models**

Latent growth models are models with continuous latent variables in which factor loadings are restricted in a specific way. Below are a few examples of growth models, which can be applied to continuous, dichotomous, or count dependent variables (the dependent variables/indicators are named ‘Item1’ – ‘Item4’ below). For ordinal and nominal dependent variables some changes would be required.

A linear latent growth model is a model with two continuous factors and is defined as follows:

Factor0 <- 1;
Factor1 <- 1;
Factor0;
Factor1;
Factor0 <-> Factor1;
Item1 <- (1) Factor0 + (0) Factor1;
Item2 <- (1) Factor0 + (1) Factor1;
Item3 <- (1) Factor0 + (2) Factor1;
Item4 <- (1) Factor0 + (3) Factor1;

Here, Factor0 represents the intercept factor and Factor1 the slope factor. Each case is thus assumed to have its own unique intercept and slope. Note that the regression equations for the
indicators do not contain intercepts, which is why the factor intercepts are identified. The factor loadings are fixed to 1 for the intercept factor and to the linear time variable for the slope factor.

A growth model with a non-linear slope can be defined as follows:

\[
\begin{align*}
\text{Factor0} & \gets 1; \\
\text{Factor1} & \gets 1; \\
\text{Factor0}; \\
\text{Factor1}; \\
\text{Factor0} & \rightarrow \text{Factor1}; \\
\text{Item1} & \gets (1) \text{Factor0} + (0) \text{Factor1}; \\
\text{Item2} & \gets (1) \text{Factor0} + (1) \text{Factor1}; \\
\text{Item3} & \gets (1) \text{Factor0} + (a) \text{Factor1}; \\
\text{Item4} & \gets (1) \text{Factor0} + (b) \text{Factor1};
\end{align*}
\]

An alternative parameterization with uncorrelated factors and factor variances equal to 1 is given by:

\[
\begin{align*}
\text{Factor0} & \gets 1; \\
\text{Factor1} & \gets 1; \\
(1) \text{Factor0}; \\
(1) \text{Factor1}; \\
\text{Item1} & \gets (a) \text{Factor0} + (b) \text{Factor1}; \\
\text{Item2} & \gets (a) \text{Factor0} + (c) \text{Factor1}; \\
\text{Item3} & \gets (a) \text{Factor0} + (d) \text{Factor1}; \\
\text{Item4} & \gets (a) \text{Factor0} + (e) \text{Factor1};
\end{align*}
\]

Whereas the linear latent growth model defined above assumed fixed locations of the four time points (at 0, 1, 2 and 3), it is also possible to define a linear growth model with varying locations of time points (for a non-balanced design). This is achieved as follows:

\[
\begin{align*}
\text{Factor0} & \gets 1; \\
\text{Factor1} & \gets 1; \\
\text{Factor0}; \\
\text{Factor1}; \\
\text{Factor0} & \rightarrow \text{Factor1}; \\
\text{Item1} & \gets (1) \text{Factor0} + (1) \text{Factor1} \text{Time1}; \\
\text{Item2} & \gets (1) \text{Factor0} + (1) \text{Factor1} \text{Time2}; \\
\text{Item3} & \gets (1) \text{Factor0} + (1) \text{Factor1} \text{Time3}; \\
\text{Item4} & \gets (1) \text{Factor0} + (1) \text{Factor1} \text{Time4};
\end{align*}
\]

where Time1 - Time4 are the independent variables containing the actual time points for each individual. In other words, to handle records containing possibly different time points across individuals, an interaction between the slope factor and the time variable is defined for the associated measurement, and the loading of this interaction term is fixed to 1. The same model
can also be defined in its mixed effects formulation (that is, without a regression model for the latent factors), yielding:

\[
\text{Factor0;}
\]
\[
\text{Factor1;}
\]
\[
\text{Factor0} \leftrightarrow \text{Factor1;}
\]
\[
\text{Item1} \leftarrow (a) \ 1 + (b) \ \text{Time1} + (1) \ \text{Factor0} + (1) \ \text{Factor1} \ \text{Time1;}
\]
\[
\text{Item2} \leftarrow (a) \ 1 + (b) \ \text{Time2} + (1) \ \text{Factor0} + (1) \ \text{Factor1} \ \text{Time2;}
\]
\[
\text{Item3} \leftarrow (a) \ 1 + (b) \ \text{Time3} + (1) \ \text{Factor0} + (1) \ \text{Factor1} \ \text{Time3;}
\]
\[
\text{Item4} \leftarrow (a) \ 1 + (b) \ \text{Time4} + (1) \ \text{Factor0} + (1) \ \text{Factor1} \ \text{Time4;}
\]

*Mixture variants* of the above latent growth models can be obtained by including a nominal latent variable (say Cluster) and adding these three equations to the model:

\[
\text{Cluster} \leftarrow 1;
\]
\[
\text{Factor0} \leftarrow 1 \mid \text{Cluster;}
\]
\[
\text{Factor1} \leftarrow 1 \mid \text{Cluster;}
\]

In the mixed effects formulation one will instead assume that the intercept and slope in the model for the response variables depends on clusters; e.g, for Item1,

\[
\text{Item1} \leftarrow (a) \ 1 \mid \text{Cluster} + (b) \ \text{Time1} \mid \text{Cluster} + (1) \ \text{Factor0} + (1) \ \text{Factor1} \ \text{Time1;}
\]

5.3.5 *Latent growth models as models for a repeated univariate response*

When information on multiple time points is contained in a single variable (e.g., Time) in different records (as in the Regression module), a latent growth model is defined as follows:

\[
\text{Factor0} \leftarrow 1;
\]
\[
\text{Factor1} \leftarrow 1;
\]
\[
\text{Item} \leftarrow (1) \ \text{Factor0} + (1) \ \text{Factor1} \ \text{Time;}
\]

or as

\[
\text{Factor0;}
\]
\[
\text{Factor1;}
\]
\[
\text{Factor0} \leftrightarrow \text{Factor1;}
\]
\[
\text{Item} \leftarrow 1 + \text{Time} + (1) \ \text{Factor0} + (1) \ \text{Factor1} \ \text{Time;}
\]

in its mixed model formulation.
5.4 Latent class regression models

Next, we provide some examples similar to the models that can be defined with the Latent GOLD Regression module, where multiple replications are available for each case.

5.4.1 Simple LC regression model

A simple unrestricted LC regression with two predictors – ‘quality’ and ‘price’ – has the following form:

```
Class <- 1;
response <- (a) 1 | Class + (b) quality | Class + (c) price | Class;
```

As can be seen, the conditional effects specification (‘| Class’) is used to indicate that the three parameter sets depend on ‘Class’. Class-independent effects are obtained by removing the conditioning on ‘Class’ from the associated term. Restricting effects in some classes to 0 involves specifying fixed value restrictions, e.g., no effect of ‘Price’ in ‘Class’ number 1:

```
c[1,] = 0;
```

Similarly, equality restrictions can be specified across classes. For example,

```
c[2,] = c[1,];
```

yields a model with price effects equal in classes 1 and 2.

This shows that all restrictions available in the GUI Regression module can also be specified with the syntax. Additional syntax restrictions that are not available in the GUI are ratio restrictions (parameter ‘a’ is twice as large as parameter ‘b’), equality restriction across predictors, and fixed value restrictions which are not 0 or 1.

Another important added capability of Syntax for LC regression analysis is the possibility to include interaction terms in the regression model. For example, a quality x price interaction may be included in a model as follows:

```
Class <- 1;
response <- (a) 1 | Class + (b) quality | Class + (c) price | Class + (d) quality price | Class;
```

5.4.2 LC regression model for two or more dependent variables

Probably the most interesting extension for LC regression modeling offered by the Syntax is the possibility to have multiple response variables. An example of such a model with two dependent variables is

```
Class <- 1;
```
response1 <- (a1) 1 | Class + (b1) quality | Class + (c1) price | Class;
response2 <- (a2) 1 | Class + (b2) quality | Class + (c2) price | Class;

where response1 and response2 are dependent variables with possibly different scale types. Many variants of this are possible, such as a variant with two associated nominal latent variables, one for each dependent variable:

Class1 <- 1;
Class2 <- 1;
Class1 <-> Class2;
response1 <- (a) 1 | Class1 + (b) quality | Class1 + (c) price | Class1;
response2 <- (a) 1 | Class2 + (b) quality | Class2 + (c) price | Class2;

Responses could be allowed to be correlated (at a particular replication), which involves including the covariance equation

response1 <-> response2;

5.4.3 LC regression model with a random intercept

A LC regression model can include a random intercept, specified with a continuous factor affecting the response variable. That is,

Class <- 1;
response <- (a) 1 | Class + (b) quality | Class + (c) price | Class + (d) Factor;

or with a free Factor variance and the factor effect fixed to 1

Factor;
Class <- 1;
response <- (a) 1 | Class + (b) quality | Class + (c) price | Class + (1) Factor;

5.5 Latent Markov models

Suppose you have a latent Markov model for a single indicator (‘response’) having 3 categories, and 2 covariates -- the nominal time-varying variable ‘time’ and the nominal time-constant variable ‘sex’. The time-specific (dynamic) 3-class latent variable is denoted by ‘MClass’. The Variables statement for this type of model would be:

variables
caseid casenr;
independent time nominal, sex nominal;
dependent response nominal;
latent MClass nominal dynamic 3;
Note that time-varying covariates affecting transitions should take on a missing value for time point 0. The reason for this is that transitions are undefined for time point 0.

Also note that in Markov modeling all records with missing values should be retained. That is, the following option should be used:

\[
\text{options}
\text{missing includeall;}
\]

The reason for this is that eliminating records with missing values would distort the structure of the data file. That is, when reading the data file we assume that all time points are present starting at T=0 to the last time point for a given case concerned, whether or not these contain missing values.

For continuous-time Markov models, one should also define the variable indicating the length of the time intervals between the measurement occasions. That is,

\[
\text{variables}
\text{timeinterval intervallength;}
\]

Below we show how syntax equations can be used to specify various special cases of the Markov model. It should be noted that the equations remain the same, irrespective of the number of time points.

5.5.1 **Time-homogeneous latent Markov model**

A simple latent Markov model with time-homogeneous transition and measurement error probabilities is obtained as follows:

\[
\begin{align*}
\text{MClass}[=0] & \sim 1; \\
\text{MClass} & \sim 1 + \text{MClass}[-1]; \\
\text{Response} & \sim 1 + \text{MClass};
\end{align*}
\]

or using conditional effects as

\[
\begin{align*}
\text{MClass}[=0] & \sim 1; \\
\text{MClass} & \sim 1 | \text{MClass}[-1]; \\
\text{Response} & \sim 1 | \text{MClass};
\end{align*}
\]

The special running-reference-category coding for the logit parameters is obtained as follows:

\[
\begin{align*}
\text{MClass}[=0] & \sim 1; \\
\text{MClass} & \sim (~\text{tra}) 1 | \text{MClass}[-1]; \\
\text{Response} & \sim (~\text{err}) 1 | \text{MClass};
\end{align*}
\]
(for the response this makes sense only if the number of classes is the same as the number of response categories). Note that in continuous-time models both the use of ~tra and the conditioning on the previous state is obligatory.

A simple Markov model without measurement error is obtained by expanding the latter specification with the restriction:

\[
\text{Response} \sim (b_{\text{err}}) 1 \mid \text{MClass};
\]

\[
b = -100;
\]

which fixes all classification error logits to –100 and the corresponding probabilities to (very close to) zero.

### 5.5.2 Latent Markov model with time-heterogeneous transitions

A latent Markov model with time-heterogeneous transitions is obtained by using the independent variable ‘time’ as a predictor in the model for the transition probabilities; that is,

\[
\text{MClass}[=0] \sim 1;
\]

\[
\text{MClass} \sim (\sim \text{tra}) 1 \mid \text{MClass}[-1] \text{ time};
\]

\[
\text{Response} \sim (\sim \text{err}) 1 \mid \text{MClass};
\]

or

\[
\text{MClass}[=0] \sim 1;
\]

\[
\text{MClass} \sim (\sim \text{tra}) 1 \mid \text{MClass}[-1] + (\sim \text{tra}) \text{ time} \mid \text{MClass}[-1];
\]

\[
\text{Response} \sim (\sim \text{err}) 1 \mid \text{MClass};
\]

which are equivalent models when ‘time’ is a nominal variable: making the intercept time dependent is the same as have an intercept and a time effect (with an identifying constraint).

### 5.5.3 Latent Markov model with a grouping variable

Similarly, you may include ‘sex’ as a grouping variable in the model, Which yields a full multiple group variant of the latent Markov model:

\[
\text{MClass}[=0] \sim 1 \mid \text{sex};
\]

\[
\text{MClass} \sim (\sim \text{tra}) 1 \mid \text{MClass}[-1] \text{ sex} + (\sim \text{tra}) \text{ time} \mid \text{MClass}[-1] \text{ sex};
\]

\[
\text{Response} \sim (\sim \text{err}) 1 \mid \text{MClass} \text{ sex};
\]

Eliminating sex from one these equations would imply that one of the three types of model probabilities – initial, transition, classification error -- does not vary between males and females.

### 5.5.4 Latent Markov model with a covariate
Instead of using ‘time’ and ‘sex’ as grouping variables defining conditional effects, we can also use them as covariates in logistic regression models for the initial state and the transitions. An example is:

\[
\begin{align*}
\text{MClass}[=0] & \leftarrow 1 + \text{sex}; \\
\text{MClass} & \leftarrow (\sim \text{tra}) 1 | \text{MClass}[-1] + (\sim \text{tra}) \text{time} | \text{MClass}[-1] + (\sim \text{tra}) \text{sex} | \text{MClass}[-1]; \\
\text{Response} & \leftarrow (\sim \text{err}) 1 | \text{MClass};
\end{align*}
\]

Here, ‘sex’ is assumed to affect (the logit of) the initial state probabilities, and ‘time’ and ‘sex’ affect (the logit of) the transition probabilities.

5.5.5 Mixture (or mixed) latent Markov model

Specification of a mixture latent Markov model requires inclusion of a second (case-specific) latent variable in the model. Say that this variable is called Class. A mixed variant of the simple latent Markov model is obtained as follows:

\[
\begin{align*}
\text{Class} & \leftarrow 1; \\
\text{MClass}[=0] & \leftarrow 1 | \text{Class}; \\
\text{MClass} & \leftarrow (\sim \text{tra}) 1 | \text{MClass}[-1] | \text{Class}; \\
\text{Response} & \leftarrow (\sim \text{err}) 1 | \text{MClass};
\end{align*}
\]

or

\[
\begin{align*}
\text{Class} & \leftarrow 1; \\
\text{MClass}[=0] & \leftarrow 1 + \text{Class}; \\
\text{MClass} & \leftarrow (\sim \text{tra}) 1 | \text{MClass}[-1] + (\sim \text{tra}) \text{Class} | \text{MClass}[-1]; \\
\text{Response} & \leftarrow (\sim \text{err}) 1 | \text{MClass};
\end{align*}
\]

A special case of this model is the Mover-Stayer model. This model is obtained with the following set of restrictions:

\[
\begin{align*}
\text{MClass} & \leftarrow (\sim \text{tra}) 1 | \text{MClass}[-1] + (\sim \text{tra}) \text{Class} | \text{MClass}[-1]; \\
\text{c}[1] = -100;
\end{align*}
\]

which constrains the transitions probabilities for the first Class to 0 (assuming ‘coding=last’).

Time-heterogeneous variants, multiple-group variants, and variants with covariates of the mixed latent Markov model can be obtained in a manner similar to that of non-mixed latent Markov models.

5.6 Multilevel LC models

Multilevel LC models are models with a ‘groupid’ and with one or more group-level latent variables.
5.6.1 Simple multilevel LC model

A simple unrestricted multilevel LC cluster model with a single discrete group-level latent variable (GClass) is defined as follows:

GClass <- 1;
Cluster <- 1 | GClass;
Item1 <- 1 + Cluster;
Item2 <- 1 + Cluster;
Item3 <- 1 + Cluster;
Item4 <- 1 + Cluster;

and a model with a group-level continuous factor (GCFactor) as:

(1) GCFactor;
Cluster <- 1 + GCFactor;
Item1 <- 1 + Cluster;
Item2 <- 1 + Cluster;
Item3 <- 1 + Cluster;
Item4 <- 1 + Cluster;

5.6.1 Multilevel LC Model with Covariates

A model with covariates in the model for the Cluster is obtained as follows:

(1) GCFactor;
Cluster <- 1 + GCFactor + sex + age;
Item1 <- 1 + Cluster;
Item2 <- 1 + Cluster;
Item3 <- 1 + Cluster;
Item4 <- 1 + Cluster;

This is a random-effects logistic regression model for an indirectly observed (or latent) nominal “response” variable.
6 Details on Various Syntax Modeling Options

This section provides model detailed information on various LG-Syntax modeling options; that is:

1. Alternative regression models for dichotomous and ordinal dependent variables
2. Alternative regression models for continuous dependent variables
3. Log-linear scale factor models for categorical dependent variables
4. Bias-adjusted step-three modeling
5. Regression models with cell weights (~wei option)
6. Continuous-time Markov models
7. The knownclass indicator
8. Using a dynamic latent variable to define an additional level in a multilevel LC model

6.1 Alternative regression models for dichotomous and ordinal dependent variables

In the Latent GOLD GUI modules, a dichotomous dependent variable is modeled using a binary logit model, by defining it to be either nominal, ordinal, or (when 0/1 coded) a binomial count. An ordinal dependent variable is modeled in the GUI using an adjacent-category ordinal logit model, which is a restricted multinomial logit model (see section 2.3 of the Technical Guide). LG syntax implements six alternative regression models for dichotomous and ordinal response variables:

1. the cumulative logit model (clogit),
2. the cumulative probit model (probit),
3. the cumulative negative log-log model (loglog1),
4. the cumulative complementary log-log model (loglog2),
5. the continuation-ratio or sequential logit model (seqlogit1), and
6. a second variant of the continuation-ratio or sequential logit model (seqlogit2)

While these models can be used for ordinal response variables, when responses are dichotomous, they reduce to the binary logit (1, 5, and 6), probit (2), and log-log models (3 and 4), respectively. Each of the four cumulative models uses a cumulative link function transforming the probability of responding in a particular category or higher into the linear predictor. The Technical Guide provides details on these models.

This is an example of a cumulative logit model:

```
variables
dependent y cumlogit;
independent time, female;
equations
y < 1 + time + female;
```
6.2 Additional regression models for continuous dependent variables

Three new regression models are implemented for specific types of continuous dependent variables. These are:

- A gamma regression model for continuous variables taking on non-negative values. The link function is log, meaning that the expected value is modeled with a linear model after a log transformation.
- A beta regression model for continuous variables taking on values between 0 and 1 (excluding these numbers). The regression model uses a logit link.
- A von Mises regression model for circular data (data indicating the position on a circle) and ranging between 0 and 2π. The regression model uses an identity link.

The Technical Guide provides detailed information on the gamma, beta, and von Mises distributions.

This is an example of a beta regression model:

```plaintext
variables
caseid id;
dependent accuracy beta;
independent dyslexia, iq;
latent Class nominal 2;
equations
Class <- 1;
accuracy <- 1 | Class + dyslexia | Class + iq | Class;
accuracy;
```

6.3 Log-linear scale factor models for categorical dependent variables

A new modeling feature is the possibility to include a scale factor in regression models for categorical response variables; that is, in models for nominal and the various types of ordinal and choice dependent variables. A scale factor is a term by which all parameters in the regression model are multiplied, and which thus allows modeling proportionality of parameter values across groups. The inverse of the scale factor is proportional to the standard deviation of the error distribution when using the underlying latent variable interpretation of the categorical response regression model concerned. Therefore, this option makes it possible to model heterogeneity in response (un)certainty.

Using scale model implies that the linear term in the regression model for the categorical dependent variable concerned is multiplied by a non-negative scale factor. The scale factor is
modeled by a log-linear equation, yielding a flexible approach for modeling its dependence on latent and/or independent variables. The Technical Guide provides more information of these log-linear models for scale factor.

This is a simple example of an ordinal regression model with a scale factor:

```plaintext
variables
daependent y cumlogit;
dependent time, female;
equations
  y <- 1 + time + female;
y <- female;
```

Here, it is assumed that males and females differ in term of response uncertainty. If the log-linear coefficient for female is negative, it means the scale factor is smaller and the response uncertainty is larger for females compared to males.

### 6.4 Bias-adjusted step-three modeling

Latent GOLD 5.0 Basic contains a new Step3 modeling submodule, which implements the bias-adjusted three-step LC analysis approach proposed by Vermunt (2010) and Bakk, Tekle, and Vermunt (2013). These types of analyses are also available in LG-Syntax 5.0, but can be used in a more general class of models. Defining a step-three analysis in Syntax involves:

- defining the posteriors along with the latent variable(s) specification,
- specifying the step3 settings in the options section of the syntax (if one wishes another setting than ’proportional ml’, which is the default),
- and defining the remaining part of the model in the usual manner.

Let us first look at the translation of the Step3 GUI models into syntax as done automatically using the ‘Generate Syntax’ menu option. A Step3-Covariate model using proportional classification and the ML adjustment is specified as follows:

```plaintext
options
  step3 proportional ml;
  output parameters standarderrors estimatedvalues;
variables
  independent z1, z2, z3;
  latent Cluster nominal posterior=(clu#1 clu#2 clu#3);
equations
  Cluster <- 1 + z1 + z2 + z3;
```

Note that there are two important differences compared to the definition of a standard LC model with covariates. The first is that we do not specify the number of latent classes, but instead use posterior=(clu#1 clu#2 clu#3) to define the names of the variables containing the posterior
membership probabilities. The other difference is that the model definition does not contain dependent variables. However, the assigned class membership will serve as a dependent variable (with known response probabilities). This variable gets the label ‘Cluster_proportional’.

The proportional-none combination can be used to obtain the scoring equation, where the independent variables are the variables used in the original (step-1) analysis. A Step3-Dependent model using modal classification and the BCH adjustment is defined as:

```plaintext
options
    step3 modal bch;
output parameters standarderrors estimatedvalues profile;
variables
    dependent y1 ordinal, y2 nominal, y3 continuous;
latent Cluster nominal posterior=(clu#1 clu#2 clu#3);
equations
    y1 <- 1 + Cluster;
    y2 <- 1 + Cluster;
    y3 <- 1 + Cluster;
```

Note that we do not include “Cluster <- 1;” in a Step3-Dependent model because the cluster sizes are automatically fixed to the average posterior membership probabilities. It should also be noted that the BCH adjustment is the preferred method with dependent variables which are continuous or counts (Bakk and Vermunt, 2013).

Now we will give several examples of more general step-three models that can be estimated with LG-Syntax 5.0. This is a model with 2 latent variables:

```plaintext
options
    step3 proportional ml;
output parameters standarderrors estimatedvalues;
variables
    independent z1, z2, z3;
latent Cluster1 nominal posterior=(clu#11 clu#12 clu#13)
    Cluster2 nominal posterior=(clu#21 clu#22);
equations
    Cluster1 <- 1 + z1 + z2 + z3;
    Cluster2 <- 1 + Cluster1 + z1 + z2 + z3;
```

An example of a step-three latent Markov model is:

```plaintext
options
    step3 modal ml;
output parameters standarderrors estimatedvalues;
variables
    caseid id;
    independent z;
```
latent State nominal dynamic posterior=(class#1 class#2);
equations
  State[=0] <- 1;
  State <- (~tra) 1 | State[-1] + (~tra) z | State[-1];

An example of a 3-step multilevel latent class model with a level-2 random effect and a level-1 covariate is:

options
  step3 modal ml;
  output parameters standarderrors estimatedvalues;
variables
  groupid teamid;
  independent z;
  latent U continuous, Class nominal posterior=(class#1 class#2);
equations
  U;
  Class <- 1 + (1) U + z;

Note that in models in which one or more id variables needs to be specified, only the modal-ml and modal-none combinations can be used. This applies to mixture regression models with replications, latent Markov models, and multilevel LC model.

6.5 Regression models with a cell weight vector (“~wei” option)

The specification “~wei” is used to include fixed parameter values in multiplicative form in models for nominal and ordinal latent or dependent variables. In the log-linear analysis literature, these are referred to as cell weights (Clogg and Eliason, 1987; Vermunt, 1997). A cell weight is equivalent to what is known as an offset in generalized linear modeling, but then in exponential form; thus, a cell weight equals exp(offset) or an offset equals log(cell weight) (Agresti, 2002). Among other uses, cell weights can be used to fix probabilities to specific values.

We use the cells weights ourselves in some GUI models. That is, to define mover-stayer models, in which transition probabilities are fixed to 0 for the stayer class, to define manifest Markov models in which one indicator is perfectly related to the classes, and to define step-three analyses using ML adjustments in which the assigned class serves as an indicator with “known” response probabilities.

This is an example of the use of the cell weight vector to define a mover-stayer model:

variables
caseid id;
dependent y nominal;
latent
  Class nominal 2,
State dynamic nominal 2;
equations
  Class <- 1;
  State[=0] <- 1 | Class;
  State <- (~tra) 1 | State[-1] + (w~wei) State[-1] | Class;
  y <- 1 + State;
  w[1] = {1 0
            0 1};

Note that we define the weight vector only for Class=1 (transition are 0 for this class). For Class-2 the weights take on their default value of 1, meaning that they do not affect the model. The example above also shows another new feature of LG-Syntax, namely the option to define the fixed values or starting values for a vector of parameters by putting them between braces “{…}”.

### 6.6 Continuous-time Markov models

LG-Syntax 5.0 contains a new option for defining continuous-time (discrete-state) latent Markov models (Jackson and Sharples, 2002; Sharples et al., 2003; Böckenholt, 2005). In these models, the transition probabilities between measurement occasions are modeled as a function of transition intensity parameters and the length of the time interval. Transition intensities are in fact equivalent to hazard rates in survival models; that is, the continuous-time equivalents of the probability of experiencing an event or transition (Cox and Miller, 1965; Kalbfleisch and Lawless, 1985).

When the data set contains observations at equidistant time intervals, one will typically use a discrete-time latent Markov model. However, in such situations, one can also use a continuous-time model, which will usually yield the same fit (same log-likelihood value), but the transition parameters take on a different interpretation. However, the continuous-time model is especially useful when observations do not occur at equidistant time intervals; that is, if the measurement occasions consist of snapshots of the underlying continuous-time process taken at arbitrary points in time. The only additional information that is needed to define a continuous-time latent Markov model in Latent GOLD 5.0 is information on the length of the time intervals between the measurement occasions. The Technical Guide provides the relevant formulae.

The specification of continuous-time latent Markov models in LG-Syntax 5.0 requires:

- the specification of a variable “timeinterval” indicating the length of the time interval between the current and the previous measurement occasion;
- the regression model for the transitions to contain only effects which are conditioned on the previous state transition and which use transition coding (indicated with ~tra); this is necessary because only the parameters for \( r \neq s \) are defined;
- and the regression model for the transitions to contain an intercept or a term labeled as intercept using “~int” (see equation options). This is necessary because the starting values for the intercept (should) depend on the time scale used for the time intervals (minutes, hours, days, etc.). The model itself is not affected by the chosen time scale.
The requirements related to the transition equation are checked, and an error message is displayed when something is incorrect.

This is an example of a continuous-time latent Markov model with 3 indicators and 1 covariate:

variables
caseid id;
timeinterval dt ime;
independent z;
latent State nominal dynamic 2;
dependent y1, y2, y3;
equations
State[=0] <- 1;
State <- (~tra) 1 | State[-1] + (~tra) z | State[-1];
y1 – y3 <- 1 | State;

6.7 Knownclass option

The ‘knownclass’ option can be used to include partial classification information for nominal and ordinal case-level latent variables. In applications where a subset of the cases are known with certainty not to belong to particular classes, a knownclass indicator can be used to restrict their posterior membership probability to 0 for these classes. Hence, these cases are classified into one of the remaining class(es) with a total probability equal to 1.

In general, a knownclass variable is followed by a series of ls and 0s for each class which, as described in more detail below, indicate to which classes cases may be assigned. However, if the knownclass variable is numeric, such case assignment information need not be specified explicitly, in which case the following default options are used to assign cases to the appropriate classes. For a latent variable containing K latent classes, cases containing a value between 1 and K on the knownclass indicator are assigned to the corresponding class with probability 1. Cases containing any other value (including missing) on the knownclass variable may be assigned to any of the K classes (i.e., no restrictions are placed upon the posterior probabilities).

More specifically, following the knownclass variable name, case assignment information may be given by specifying particular values taken on by the knownclass variable followed by a sequence of K 1’s or 0’s indicating whether or not cases with the specified code may be assigned to a certain class of the latent variable concerned. We now illustrate this with a few examples.

Example 1: knownclass indicator is a numeric variable

- latent Cluster nominal 3  knownclass=TRUE(1: 1 0 0, 2: 0 1 1, 3: 0 0 1);

Here, cases coded TRUE = 1, are assigned to Cluster = 1, those coded TRUE = 2 may be assigned to either Cluster 2 or 3, those coded TRUE = 3 are assigned to Cluster 3 and the posterior probabilities for all other cases (if any) are unrestricted – they may be assigned to any of the 3 classes.
Example 2: knownclass indicator is a (character) string variable

- latent Cluster nominal 3 knownclass=TRUE('a': 1 0 0, 'b': 0 1 1, 'c': 0 0 1);

Here, the assignments are as explained above except that TRUE is a string variable with codes 'a', 'b', and 'c'.

Example 3:

- latent Cluster nominal 3 knownclass=Gender('Female': 1 0 0, 'Male': 0 1 1);

Here, the assignments are similar to that in Example 2 where cases coded 'Female' are restricted to the first class, and cases coded 'Male' are restricted to the second and third class. The posterior probabilities for cases coded in some other way (or containing a missing value) remain unrestricted, so that these cases are free to be assigned to any of the three classes.

When generating the syntax from a GUI model which includes a knownclass indicator, you should use the ‘scan’ or ‘estimate’ option prior to generating your syntax. This is needed so the program knows what values the knownclass variable takes on. If neither of these options are executed prior to attempting to generate the syntax you will receive an error message as follows:

6.8 Using a dynamic latent variable to define an additional level in a multilevel LC model

The LG Syntax can be used to define multilevel models with group- and case-level latent variables. However, it is also possible to define models with latent variables at three-levels using the dynamic option. More specifically, if a latent variable is defined to be dynamic and the equations do not contain the time point 0, [=0], and the previous time point, [-1], versions of this variable, the latent variable concerned is assumed to be a lower-level latent variable; that is, a latent variable for observations nested within the units denoted by the caseid. An example of such a model is:

```
variables
groupid group;
caseid person;
dependent y1, y2, y3;
independent z1 nominal, z2;
```
latent T group nominal 2, X nominal 2, W nominal dynamic 2;
equations
T<-1;
X <- 1 | T + z1 | T;
W <- 1 | X + z2;
y1 - y3 <- 1 | W;
7 Details on Various Syntax Tools and Output Options

This section provides more details on the following features of the LG-Syntax that require further explanation:

1. The structure of internal parameters
2. Obtaining additional output and scoring external files
3. The multiple imputation of missing values option and the option for analyzing multiply imputed data sets
4. The various simulation features, which include the ability to simulate multiple data sets, analyze multiple simulated data sets, perform fully automated Monte Carlo simulation studies, and to perform power calculations using MC simulation
5. User-defined Wald tests
6. Power computation for Wald tests
7. Score tests and EPCs
8. Alternative complex sampling options
9. Identification checking
10. Validation and hold-out options
11. Options to write output to text files

7.1 Internal parameter structure

Each of the parameters of a LG-Syntax model has an internal number associated with it. This number can be displayed in the Parameters output section by activating the column ‘Internal’ using the popup menu. Figure 6-1 shows an example of the Parameters output with this ‘Internal’ column for a model containing 9 free parameters which are enumerated from 1 to 9. The redundant parameters (in this case for the last category of the nominal variables Class, year and religion) are not estimated and thus do not have any number associated with them.

![Figure 6-1: Parameters with ‘Internal’ column.](image-url)
Using the option ‘Syntax with Parameters’ in the File Save Syntax menu command, the saved syntax for this example model contains the following Equations Section:

```plaintext
equations
  response <- 1 | Class + year + religion ;
  Class <- 1;
  {2.224168744663137
    0.07198244434239946
    0.1115934817516651
    -0.3844805646738486
    0.01005171849917819
    -0.3747469959227842
    0.2806252836538088
    -0.2935911967842039
    -0.1721545534454281
  }

The numbers between ‘{}’ are the saved parameter values. Comparing these numbers to those given in the ‘Coef’ column (see Figure 6-1) and it can easily be verified that the order in which these are saved (and restored) corresponds to the internal numbering as displayed in the column labeled ‘Internal.’ In addition, the columns labeled ‘term’ provide a description of these parameters which describe what each of these numbers is (e.g., the number 2.224168744663137 is the intercept for the dependent variable ‘response’ for the first class).

When running a model with parameter values specified between ‘{}’, these values will be used as starting values. This means that reopening and rerunning a syntax model that was saved with the estimated parameters will reproduce the original output without the need of performing EM or Newton-Raphson iterations.

When saving a model with parameters, ‘{}’ is added automatically to the syntax file. However, it is also possible to use this option to specify starting values for all parameters directly. This is a very convenient way to provide starting values, and may also be used for defining the population values in MC simulation studies (see below). The program will check whether the number of parameters provided between ‘{}’ corresponds to the number of free parameters in the specified model. As a check of the correctness of the starting values specification, you may request that model be run without any iterations being performed by using the options ‘emiterations=0’ and ‘nriterations=0’, and inspect the Parameters output as to whether the starting values match the intended parameters.

In most cases, the internal parameters between braces correspond to the parameters reported in the Parameters output. There is, however, one exception, which occurs when monotonicity constraints are imposed. In that case, because of an internal reparameterization of the model, the internal parameters will reflect differences between categories rather then the reported parameters themselves. An example is provided in Figure 6-2, which shows the result of running the same model as above but now with a monotonic effect of ‘year’.
The saved syntax with Parameters looks as follows:

```
equations
  response <- 1 | Class + (+) year + religion;
  Class <- 1;
{
  2.214729705811549
  0.0731352361289665
  0
  0.1521609438138654
  0.2523882920970541
  -0.3732746339509661
  0.297925487051707
  -0.2913124993142653
  -0.173720343880652
  }
```

As can be seen, parameter numbers 3, 4, and 5 within the ‘{}’ do not correspond to the values appearing in the Parameters output, but instead to the difference between the next and the current category (e.g., 0.1522 = 0.0130 - 0.1392).

Another issue that deserves special attention is reparameterization of the (co)variances of the latent variables when numerical integration is used. In this situation, the internal parameters are not (co)variances, but elements of the Cholesky decomposed (co)variance matrix. These are reported as parameters in the Parameters output with an appendix ‘(chol)’. The lower part of the Parameters output reports the variances and covariances, which are obtained from these internal parameters, but which are not internal parameters themselves.
Figure 6-3: Cholesky parameters and Variances/Covariances of Continuous Latent Variables

Figure 6-3 gives an example the Parameters output for a latent growth model with two latent factors and a dichotomous dependent variable. As can be seen, the sections ‘Variances’ and ‘Covariances/Associations’ contain parameters with an appendix ‘(chol)’. These parameters have an internal number and are thus part of the set of internal model parameters. The bottom of Figure 6-3 shows the ‘Variances / Covariances continuous latent’, which is what will typically be of interest. As can be seen, these parameters do not have an internal number.

To illustrate the relation between these two sets of parameters, note that the estimated covariance matrix in the above example equals

\[
\begin{pmatrix}
7.0617 & -2.1242 \\
-2.1242 & 2.9837
\end{pmatrix}
\]

and that lower-diagonal Cholesky matrix equals

\[
\begin{pmatrix}
2.6574 & 0 \\
-0.7994 & 1.5312
\end{pmatrix}
\]

The covariance matrix equals the Cholesky matrix times its transpose

\[
\begin{pmatrix}
2.6574 & -0.7994 \\
0 & 1.5312
\end{pmatrix}
\]

that is, 7.0617=2.6574*2.6574, -2.1242=2.6574*-0.7994; and 2.9837=-0.7994*-0.7994+1.5312*1.5312.
7.2 Obtaining additional output and scoring external files

After estimating a model, additional output statistics beyond what was already produced as well as output to a file may be obtained without re-estimating the model. The following steps should be followed to obtain additional output:

1. From the File menu, select ‘SAVE SYNTAX’
2. In the ‘SAVE CONTENTS’ drop down box at the bottom of the ‘SAVE AS’ window, select ‘Syntax with Parameters’. Assign a file name and select 'Save'.
3. From the File menu, select Open to open the saved ‘lgs’ file in the Syntax Editor.
4. Use the Syntax editor to insert additional keywords to the output subsection, requesting the additional output, or insert an outfile section to write information to an output file.
5. Select ‘Estimate’ from the Model menu.

The requested output will be produced without the model being re-estimated.

Scoring an external file works in a similar way:

1. Estimate your model in the usual way -- you may use the LG5.0 GUI or syntax to do this. From the File menu, select ‘SAVE SYNTAX’
2. In the ‘SAVE CONTENTS’ drop down box at the bottom of the ‘SAVE AS’ window, select ‘Syntax with Parameters and Variable Definitions’ (see section 2.1).
3. Assign a file name and select 'Save'.
4. Use a text editor to edit the ‘lgs’ file that was created. Change the name of the infile data file on line 4 to the name of the new file that you want to score. Also, in the output section, prior to the terminating ‘;’, insert the keyword ‘outfile’, followed by the name of the output file to be created followed by the keywords that describe the output you wish to create -- ‘classification’, ‘prediction’, etc. as indicated on section 4.1.10. For example, assuming that you are using SPSS files and you want to add the classification information to the new file, you should insert the following:

   outfile 'outfilename.sav' classification;

   where 'outfilename.sav' is the name of the new scored file to be created.

5. From the File menu of Latent Gold, select Open to open the saved ‘lgs’ file in the Syntax Editor. Select Estimate from the Model Menu. Note that in this case to re-estimate this model (on new data) takes little time since no iterations are performed.

In step 2, we selected ‘Syntax with Parameters and Variable Definitions’ instead of ‘Syntax with Parameters’. The difference is that the former adds information on the categories of the nominal and ordinal variables in the estimation data file to the saved syntax file. This information is required to be able to check whether the external data file has the same structure as the estimation data file, and if not to repair the differences by treating the non-existing categories in the file that should be scored as missing values.
7.3 Multiple imputation of missing values

LG-Syntax implements an option to impute missing values on the variables specified as dependent variables in any model, as well as an option to analyze multiply imputed data files. Vermunt, Van Ginkel, Van der Ark and Sijtsma (2008) provide the theoretical foundation of imputation based on latent class models. The appendix of this article provides examples of LG-Syntax files.

Suppose a simple latent class model is estimated and you wish to generate 10 complete data sets by multiple imputation from this model. This is achieved by means of the ‘imputation’ option in the outfile statement. An example is

    outfile 'complete10.sav' imputation=10;

This will create 10 versions of the original data file with the missing values imputed. The 10 versions are stacked in a single data file, where a newly created variable ‘imputation_#' indicates the data file number.

⚠️ A data set to be multiply imputed should not be grouped data; that is, no groupweight, caseweight, or replicationweight should be used. Moreover, the model used for multiple imputation should be run with the missing values on the dependent variables included.

To take into account parameter uncertainty, each imputation uses a different set of parameters from a non-parametric bootstrap procedure. This yields what Rubin called ‘proper’ multiple imputation. As an alternative, one may base each of the multiple imputations on the ML estimates (and thus ignore parameter uncertainty). This faster procedure is invoked by adding the command ‘EM’ after ‘imputation=’.

When applying the multiple imputation procedure to data sets with large numbers of variables, the imputation model will typically contain a huge number (hundreds) of parameters. To reduce the memory requirements and the chance of being unable to estimate the model due to insufficient memory in such situations, we recommend that you run a (potential) imputation model with all output options off (including parameters and standard errors). Moreover, because Newton-type algorithms are not well behaved in models with hundreds of parameters, one should also make sure that the program uses only the EM algorithm. This is achieved by setting nriterations=0 and emiteration=5000 to assure that a sufficiently large number of EM iterations are performed. Once an imputation model is selected, one can save the model with the estimated parameters (see previous two subsections), reopen the saved model, add the outfile line for generating multiple imputations, and rerun. This will yield the requested number of completed data sets.

Not only can LG-Syntax be used to generate multiple imputations, it can also be used to analyze multiply imputed data sets, where the multiple imputations may, of course, also be obtained with specialized multiple imputation software. There are only two differences compared to running LG-Syntax for the analysis of a single data set: (1) the data file should consist of n stacked data
sets with an id variable indicating the data set number, and (2) the statement ‘imputationid <variable name>’ should be included in the Variables Section of the syntax file. For example,

    imputationid imputation_;

The program will analyze each of the n data sets separately and combine the results using the formulae provided by Donald Rubin: the reported parameter estimates are averages across the n analyzed data sets and variances are averages plus a term that depends on the between data sets differences in parameter estimates. This is done for Parameters, Profile, EstimatedValues-Model, and ProbMeans-Model; i.e., all output sections for which standard errors are reported. The Statistics and the other output sections are based on the average values of the internal parameters.

The Model output contains an additional section ‘Multiple Imputation Statistics” which reports the average L-squared and Log-likelihood values across the multiple data sets, an F test for the L-squared, and the missing information effect, which is the overall proportional increase of the variance caused by differences between the data sets. The Parameters output will report t tests instead of z tests and F tests instead of Wald tests.

### 7.4 Monte Carlo simulation and power calculation features

LG-Syntax implements the following simulation options:

1) the ‘outfile '<filename>' simulation=,%d’ option is used to simulate multiple (%d) data sets according to a user-specified population model, which can be combined with the ‘simulationid <variable name>' option to analyze multiple simulated data sets,

2) the ‘MCstudy[=’model name’]’ option in the ‘montecarlo’ subsection to perform a simulation study without the need to save the generated data sets to output files, and

3) the ‘power=’model name’’ option in the ‘montecarlo’ subsection, which performs a power calculation for the likelihood-ratio test for a set of parameters by means of simulation.

Irrespective of which of these options is used, simulating data sets involves (1) defining a population model, and (2) providing an ‘example’ data file. A population model is a LG-Syntax model containing “starting values” for all free (internal) model parameters. The easiest way to provide the starting values is via the braces ‘{}’ option described above. Thus a model definition such as:

    equations
    response <- 1 | Class + year + religion ;
    Class <- 1;
    {2.0 0.0 0.0 -0.5 0.0 -0.4 0.3 -0.3 -0.3}

could serve as a population model.

The example data file should provide information on the group structure in multilevel models, the number of time points in Markov models, and the number of replications for each case when
there are multiple observations per case (or per time point). In addition, the values of the
independent variables should be specified. The dependent variables (which will be simulated)
may take on arbitrary but admissible values. Even though it is not absolutely necessary, for
compactness, an example data file will typically contain group-, case, and/or replication weights
indicating how many observations of each type should be simulated. When using the ‘MCstudy’
and ‘power’ options it is also possible to specify the total sample size (at the highest level) using
the option ‘N=%d’ in the ‘montecarlo’ subsection.

Suppose you want to simulate a data sets containing 3 continuous dependent variables ‘y1’,
‘y2’, and ‘y3’, and one dichotomous predictor ‘treatment’ (1=treatment; 2=control). An example
data file could have the following form:

treatment y1 y2 y3 freq
1 0 0 0 150
2 0 0 0 150

This will generate a data set with a total of 300 observations, 150 of which belong to the
treatment and 150 to the control group. When using the ‘MCstudy’ or ‘power’ options, the same
results can also be achieved using

treatment y1 y2 y3
1 0 0 0
2 0 0 0

combined with the command ‘N=300’ in the montecarlo subsection.

An example of a possible population model for this example is

variables
  caseweight freq;
  dependent y1 continuous, y2 continuous, y3 continuous;
  independent treatment nominal coding=last;
  latent Cluster nominal 3 coding=last;
equations
  Cluster <- (a) 1 + (b) treatment;
  y1 <- (c) 1 | Cluster;
  y2 <- (d) 1 | Cluster;
  y3 <- (e) 1 | Cluster;
  (s) y1;
  (s) y2;
  (s) y3;
{-1.09861 -1.09861
  2.197225 1.09861
  60 60 60
  50 50 60
  50 60 60

83
This is a 3-class latent growth models for three measurement occasions (y1, y2, and y3). Class membership depends on the treatment, where the treated group is much more likely to belong to class 1 instead of 3 and slightly more likely to belong to class 2 instead of 3. Parameters 5 - 13 provide the means for the three growth classes, and the last parameter is the residual variance, which is assumed to be equal across classes and time points.

In some situations, a slightly more complex example data file is needed, for example, to simulate data from a latent Markov model. An example of this is

```
caseid female time y freq
1 0 . 0 500
1 0 1 0 500
1 0 2 0 500
1 0 3 0 500
1 0 4 0 500
2 1 . 0 500
2 1 1 0 500
2 1 2 0 500
2 1 3 0 500
2 1 4 0 500
```

which could be an example data file for a five-occasion latent Markov model, with transition probabilities depending on time and gender. By using ‘freq’ as a ‘caseweight’ one obtains simulated data sets 1000 observations. Again, when using the ‘MCstudy’ or ‘power’ option combined with ‘N=1000’, a ‘caseweight’ is not required.

Special caution is needed when dependent variables are nominal or ordinal (including the cumlogit, probit, loglog and seqlogit types). In such cases, it is important to inform the program about the number of categories of the dependent variables. This can be either done by making sure that all possible categories appear in the example data file or by specifying the number of categories for nominal/ordinal dependent variables as part of the variables definition (as required for nominal/ordinal latent variables).

We will now provide more detail on each of the three simulation methods. The option ‘outfile <file name> simulation[=\%d]’ will generate ‘\%d’ data sets from the specified population model. The resulting stacked data set can be analyzed by including the command ‘simulationid <varname>’ in the Variables Section (note that this is similar to what we discussed above for imputation). There is no need that the analysis model be the same type of model as the population model. The program will analyze each of the replicate data sets and subsequently combine the results. It will report means and standard deviations across replicates as parameter estimates and standard errors. This is done for Parameters, Profile, ProbMeans-Model, and EstimatedValues-Model. Also the reported statistics are averages across replicates. It is also possible to write the statistics and parameters estimates for all replicates to a compact output file using the ‘write’ or ‘append’ output options.
The option ‘MCstudy’ can be used to simulate data from a particular population model and estimate the same model when the population and analysis models are specified to be the same. More generally, ‘MCstudy=’modelname” generates data sets from the population defined in model 'modelname' and uses the current model as the analysis model. The requirement is that the two models contain the same set of dependent and independent variables. As described above for the other simulation method, the program combines the results obtained with the different replicates and reports means and standard deviations across replicates as parameter estimates and standard errors, and statistics are averages across replicates. Statistics and estimates of all replicate runs can be written to a compact output file using the ‘write’ or ‘append’ output options.

The command ‘power=’modelname' can be used to generate data sets from the population defined in model 'modelname' and estimate both the population and the current model. In power studies, the population model is the unrestricted (or H1) model and the current model is the restricted (or H0) model. When implemented by means of monte carlo simulation, power is defined as the proportion of replicates for which H0 is rejected in favor of H1 using a likelihood-ratio test. The power estimate is the only relevant output when running a model with the ‘power=’modelname” option.

As an illustration of power calculation by means of simulation, suppose the 3-class latent class growth defined above is the population model (it is called ‘H1’), and that we want to access the power of the likelihood-ratio test for the two parameters labeled ‘b’; that is, for the effect of treatment on growth class membership. In other words we want to assess the power of a test of whether treatment increases the likelihood of having a more change pattern (of belonging to a class with a decreasing mean). The relevant Options and Equations sections of the H0 model are

```plaintext
options
    montecarlo power='H1' and N=300;

equations
    Cluster <- (a) 1 + (b) treatment;
    Y1 <- (c) 1 | Cluster;
    Y2 <- (d) 1 | Cluster;
    Y3 <- (e) 1 | Cluster;
(s) Y1;
(s) Y2;
(s) Y3;
b = 0;
```

These statements will yield a power calculation for a sample size of 300, where the current model is compared with the model called H1. As in the population model, starting values may be provided for the free model parameters (in this model ‘b’ is not a free parameter, of course, since it is restricted to 0, and thus no starting value is provided for it). Typically, the power calculation will be repeated for various sample sizes and various effect sizes (note that in the example the assumed effects correspond to odds-ratios of $\exp[2.197225]=9$ and $\exp[1.09861]=3$).
The other three ‘montecarlo’ options in LG-Syntax are ‘L2’, ‘allchi2’, and ‘LLdiff=modelname’. These commands invoke two types of parametric bootstrap procedures which are in fact also Monte Carlo simulation procedures. While these are similar to the MCstudy and power procedures described above, an important difference is that for a parametric bootstrap you do not need to define a population since it is derived from the analyzed data set (the parameter estimates serve as population values). The purpose of the parametric bootstrap procedure is also different, namely estimating a p-value associated with a likelihood-ratio test. Note that the parametric bootstrap can also be used for computing standard errors (‘standarderrors=pbootstrap’).

7.5 **User-defined Wald tests**

It is now possible to perform user-defined Wald tests based on linear constraints on the parameters. The constraints to be tested are specified in a separate text file. The output command inducing the user-defined Wald tests is “WaldTest=’filename’.

A linear constraint in the text file has the following form:

\{(c1) p1 + (c2) p2 + … + (ck) pk = (d)\}

where c1, c2, ck, and d are constants and p1, p2, and pk are (internal) parameter numbers. Note that each constraint is enclosed within braces “\{…\}”.

Examples are “\{(1) 6 = (2)\}” and “\{(1) 3 + (-1) 2 = (0)\}”, indicating that the 6th parameter equals 2 and that parameters 2 and 3 are equal. These two restrictions can also be formulated in a more compact way using the fact that the constant can be omitted (default is (1)), that it is allowed to use a minus instead of a plus sign, and that the “= (0)” is the default. This yields “\{6 = (2)\}” and “\{3 - 2\}”, respectively.

A user-defined Wald test may concern multiple simultaneous restrictions and, moreover, one may wish to perform more than a single Wald test. This is achieved by concatenating the multiple restrictions belonging to the same Wald test and separating the different Wald test by semi-colons “;”. This is an example with 3 Wald tests, having 3, 2, and 4 restrictions, respectively:

\{r1\} \{r2\} \{r3\}; \{r4\} \{r5\}; \{r6\} \{r7\} \{r8\} \{r9\};

Here, r1, r2, etc. represent the 9 restrictions.

7.6 **Power computation for Wald tests**

In LG-Syntax 5.0, it possible to perform power computations for the Wald tests, including the user-defined Wald test discussed above (Gudicha, Tekle, and Vermunt, 2013). The setup is very similar to the one of a simulation study, and works as follows:
Open an “example” data set; that is, a data set with the structure of the data one is interested in;
Specify the assumed population model;
Include the command “WaldPower=<number>” in the list of output option.

When the specified number is between 0 and 1, the program reports the required sample size for the specified power, and when the specified number is larger than 1, the program reports the power achieved with the specified sample size.

### 7.7 Score tests and EPCs

LG-Syntax 5.0 implements a new powerful tool for testing parameter constraints; that is, it reports a score test for each restriction and the expected parameter changes when a restriction is removed. This output can be requested with the output command “ScoreTest”. In that case, one obtains:

- A score test in the Parameters output for each restricted parameter set (Oberski, van Kollenburg, and Vermunt, 2013). Score tests are also called Lagrange multiplier tests or modification indices. Note that a score test is the estimated decrease of minus twice the log-likelihood value when relaxing the constraint of interest.
- An EPC(self) value for each restricted parameter in the Parameters output (Oberski and Vermunt, 2013a). By EPC(self) we refer to the expected changes in the parameters of the restricted set itself after removing the restriction concerned.
- A new output section nested within Parameters containing the EPC(other) values for all free parameters (Oberski, 2013; Oberski and Vermunt, 2013b). These are the expected parameter changes in the other parameters when the restriction concerned is removed. EPC(other) statistics can be used for sensitivity analyses; that is, to determine whether a restriction affects the parameters of primary interest. We therefore also refer to these statistics as EPC(interest).

Score tests and EPCs can be requested in combination with various types of variance estimators (standard, robust, fast, and expected). For more information on this, see Oberski and Vermunt (2013a).

### 7.8 Alternative standard error estimators

The options for standard error estimation with complex sampling designs are expanded in LG-Syntax beyond the capabilities of Latent GOLD Advanced. Specifically, in addition to obtaining complex sampling standard errors based on the Taylor linearization estimator, LG-Syntax implements three alternative methods: jackknife, nonparametric bootstrap, and replicate weights. These methods involve reestimating the model with a series of replicate samples. In the jackknife procedure, the replicates are obtained by leaving out one PSU, and in a nonparametric bootstrap
by sampling PSUs with replacement from each stratum. The last option can be use when the survey provides replicate weights to account for the sampling design.

The jackknife and nonparametric bootstrap can also be used without complex sampling. In the jackknife procedure, the replicates will then be obtained by leaving out one observation, and in the nonparametric bootstrap by sampling observations with replacement from the total data set.

New in LG-Syntax 5.0 is the possibility to obtain standard errors based on the expected information matrix. Note that this requires processing all possible data patterns, which is feasible only for frequency tables with not too many cells. In other situations, one may request the Monte Carlo version of the expected information matrix, which is obtained by simulating a large number of observations from the population defined by the model estimates.

### 7.9 Identification checking

Certain latent class and mixture models for categorical dependent variables and/or binomial counts may not be identified. This means that the solution is not unique in the sense that different combinations of parameter values yield the same log-likelihood value, or equivalently, the same estimated probabilities. LG-Syntax contains a procedure to detect whether there are such identification problems. This procedure is activated by including the command ‘identification[=%d]’ in the list of the output options.

The identification check implemented in LG-Syntax makes use of the Jacobian matrix ($J$). A model is locally identified if the rank of the Jacobian matrix equals the number of free model parameters. Conversely, if the rank is lower than the number of free model parameters, the model is not identified. Note that we used the term local identification, which refers to the fact that conditional on a particular set of parameters there is no other set in the direct neighborhood that gives the same model probabilities.

LG-Syntax computes the Jacobian and evaluates its rank multiple times (the default is 10 times) using different random parameter values. The reported ‘number of non-identified parameters’ is the rank deficiency that is encountered most of the time. The ‘Iterationdetails’ output listing reports the rank deficiency for each trial, from which it can be seen that typically most or all trials give the same rank deficiencies.

It should be noted that this procedure does not prove global identification, but instead checks an important necessary condition for global identification, which is local identification for certain sets of parameters. In practice, in turns out that, except for computational precision issues, it does not make a difference which parameter values are used to determine the rank of the Jacobian. More specifically, if a model contains two non-identified parameters, the rank of the Jacobian will be 2 less than the number of parameters regardless of the parameter values used to evaluate the Jacobian.

Table 7-1 reports the results obtained when applying the LG-Syntax identification check to various latent class model for nominal dependent variables. This table shows several well known
identification problems, such as in the 2-class model for 2 dependent variables and the 3-class model for 4 dichotomous dependent variables. It is especially interesting to note the minimal requirements for the number of dependent variables and their number of categories in order to obtain an identified LC model. For a 2-class model to be identifiable, at least 3 dependent variables are required, whereas 3-, 4- and 5-class models require five indicators. The latter three models, however, can also be identified with three or four indicators, if the number of indicator categories fulfill certain requirements. For example, a 4-class model for three dependent variables is identified if two of the dependent variable have at least 3 categories and the other one at least 4 categories.
Table 7-1: Number of non-identified parameters in 2 to 5 class models estimated with 2 to 5 nominal dependent variables

<table>
<thead>
<tr>
<th>Number of Classes</th>
<th>Number of Dependent Variables</th>
<th>Number of Categories</th>
<th>Number of nonidentified Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>any</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>&gt;=3</td>
<td>any</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>both 2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>one &gt;=3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>all 2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>one &gt;=3, two 2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>two &gt;=3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>all 2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>one &gt;=3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>&gt;=5</td>
<td>any</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>both 2</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>one 3, one 2</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>both 3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>one &gt;=4</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>two or three 2</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>two &gt;=3, one 2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>all 3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>two &gt;=3, one &gt;=4</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>all 2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>one &gt;=3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>&gt;=5</td>
<td>any</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>both 2</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>both 3</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>one &gt;=5</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>all 2</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>all 3</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>all &gt;=4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>all 2</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>three &gt;=3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>one &gt;=3 and one &gt;=4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>&gt;=5</td>
<td>any</td>
<td>0</td>
</tr>
</tbody>
</table>
7.10 Validation and hold-out options

LG-Syntax implements three commands that can be useful for validating and estimating models using cases and/or replications excluded from the estimation process. The ‘validationid’ and ‘validationLL’ options can be used to obtain log-likelihood, classification, and prediction statistics based on n-fold validation, ‘holdout cases’ to obtain chi-squared, log-likelihood, classification, and prediction statistics for hold-out cases, and ‘holdout replications’ to obtain prediction statistics for hold-out replications.

N-fold validation is a procedure in which the sample is split into n parts (into n folds). The specified model is estimated n times, where each time one of the folds is excluded from the analysis. The idea is to compute relevant statistics for the excluded fold using the parameter values obtained with the other n-1 folds. This is repeated fold by fold, where the n statistics are aggregated. LG-Syntax program provides log-likelihood, classification and prediction statistics based on this procedure.

The ‘validationid <varname>’ option in the ‘variables’ section can be used to specify your own split of the sample. The variable <varname> contains integer values defining the n folds. The program will determine the number of folds from the number of categories of this variable. The ‘validationLL=n’ option creates the requested number of folds by assigning cases randomly to n folds of the same size.

The ‘holdout cases <varname> <list of values>’ option allows you to hold out cases from the analysis. The data file is then split into two parts, only one of which is used for parameter estimation (the one with the non hold-out cases). However, statistics are also reported for the hold-out cases.

The ‘holdout replications <varname> <list of values>’ option allows you to hold out replications from the analysis. More specifically, for each case you indicate which replications should be omitted during parameter estimation. However, prediction statistics are also reported for the hold-out replications, where for posterior and HB-like prediction, the posteriors are based on the non hold-out replications of a case.

Note that n-fold validation involves holding out complete cases from the analysis: that is, if a case is in the excluded fold all its replications are excluded from the analysis. In multilevel models, the folds are formed by groups instead of cases. In contrast, the holdout replications procedure can be used to exclude a portion of the replications within cases to be excluded during model estimation rather than entire cases. To hold out an entire case you must use the holdout cases option.
7.11 Options to write output to text files

LG-Syntax contains various options to write output to text files. These options were implemented mainly for our own research, such as to perform simulation studies and to facilitate the implementation of new options, but we are sharing these with our users in version 5.0. In this subsection, we describe the most important write options, which are:

- `write/append='filename'`,
- `WriteExemplaryData='filename'`,
- `WriteParameters='filename'`,
- `WriteHessian='filename'`,
- `WriteGradient='filename'`, and
- `WriteBCH='filename'`, `WriteClassificationErrors='filename'`, and `WriteClassificationErrorsProportional='filename'`

The commands `write='filename'` and `append='filename'` can be used to obtain a large portion of the LG-Syntax output in a compact comma delimited file. The option `write` creates a new file (or overwrites the existing file with the specified name) to output the requested information, whereas `append` adds the information an existing output file (or creates the file if it does not exist). The latter option can thus be used to obtain the output from multiple runs in a single file. When using `write/append` with one of the Monte Carlo options (e.g., `MCstudy` or `bootstrap`), the program outputs the requested information for each replication. However, this can be suppressed with the option `nowrite`, in which case only the final results appear the compact output file.

Depending on the requested output and the type of run, the output file contains:

- Warnings
- Statistics: log-likelihood, log-prior, log-posterior, X2, L2, CR2, DI, and Total BVR (and the -2LL difference)
- Bootstrap p values: for chi-squared statistics and the -2LL difference test, MC-based power for -2LL difference, and critical values for bootstrap, power, and MCstudy
- Classification Statistics
- Wald values (or Wald power information)
- Score tests and EPCs
- Bivariate Residuals
- Bootstrap p values for BVRs
- Parameters
- Profile
- EstimatedValues
- ProbMeans-Model
- Parameters SEs
- Profile SEs
- EstimatedValues SEs
- ProbMeans-Model SEs
Each of the above items appears on a separate line. The first columns of the output file contain provide the unique identifier for the run, the replication number (for MC options), the item number, and the name of the item (e.g. “Parameters”). The output file can easily be opened with Excel, R, SPSS, or another package for further processing.

The command WriteExemplaryData='filename’ creates a so-called exemplary data file. This is a data file containing all possible data patterns and frequencies equal to the expected frequencies under the model. This output can be requested in models for categorical responses, both for wide and long format data files. This file can be used among other purposes for asymptotic power computation for likelihood-ratio tests. Another application is if one wishes to see the estimated frequencies for all possible data patterns, and not just the observed data pattern (as reported in Frequencies).

The command WriteParameters='filename’ writes the parameter values to a file. This can be useful if one wishes to use the parameters, for example, as starting values, in another run. Note that starting values can be read from a file.

The command WriteHessian='filename’ yields the negative Hessian, its inverse, the parameters’variance-covariance matrix, and the gradients of the specified model. The negative Hessian is sometimes needed for specific computations. Note that the parameters’ variance-covariance matrix equals the inverse negative Hessian when using “standard” SEs estimates. The gradients are of interest, for example, in Score tests, in which one needs the gradients of the fixed parameters.

The command WriteGradient='filename’ yields the gradient contributions per observation.

The command WriteBCH='filename’ creates an expanded data file with one record per latent class for each observation and the weights needed for the various types of non-adjusted and bch-adjusted 3-step analysis methods. WriteClassificationErrors='filename’ and WriteClassificationErrorsProportional='filename’ provide the classification error matrix (the D matrix) on a logit scale, as well as the information needed for the first- and second-order corrections of the step-three standard errors (Bakk, Oberski and Vermunt, 2013). We used these options for our research of three-step LC analysis and during the implementation of the new step-three modeling options in Latent GOLD 5.0.

Aside of the above output to file options, there is another new output file option which is related to the scoring equation. With WriteSPSSSyntax='filename’ or WriteGenSyntax='filename’, one can create an SPSS or generic (C code) syntax file to score new cases. This requires using the step3-analysis option with independent variables affecting the class membership. In a scoring model, these independent variables are the variables used in the original analysis.
8 Getting Started with LG-Syntax

8.1 The application and the data

DemoData = 'conjoint.sav'
This tutorial introduces the use of the LG-Syntax module, an add-on to the Advanced version of Latent GOLD. In this tutorial we utilize the data which was also used in ‘Tutorial #3: LC Regression with Repeated Measures.’

The Goal
Since it is quite easy to setup a GUI model in Latent GOLD using the LG5.0 Windows Menu system, it is often useful to begin with a GUI model containing the basic elements of the desired syntax model. This GUI model can then be converted to an initial syntax model automatically using the ‘Generate Syntax’ option from the ‘Model’ menu. The goal of this tutorial is to illustrate this process, as well as show how to modify LG-Equations to obtain additional models.

In this tutorial we will:

➢ Introduce the use of LG-Syntax

➢ Show how the LG-Syntax can be generated from a GUI model

➢ Examine the Equations section of the LG-Syntax

➢ Modify the LG-Equations to specify a different LC regression model

➢ See how parameter restrictions may be specified in different ways using the syntax

We will reuse the data from ‘Tutorial 3: LC regression with Repeated Measures’ in this tutorial. While the data were generated under the assumption of the ordinal logit model, for simplicity in introducing the equation section of the LG-Syntax we will treat the dependent variable as continuous rather than ordinal, so that the models obtained are LC (linear) regression models.

The Data
The data for this example are obtained from a hypothetical conjoint marketing study involving repeated measures where respondents were asked to provide likelihood of purchase ratings under each of several different scenarios. A partial listing of the data is shown in Figure 8-1.
As suggested in Figure 8-1, there are 8 records for each case (there are 400 cases in total); one record for each cell in this 2x2x2 complete factorial design of different scenarios for the purchase of a product:

- **FASHION** (1 = Traditional; 2 = Modern)
- **QUALITY** (1 = Low; 2 = High)
- **PRICE** (1 = Lower; 2 = Higher)

The dependent variable (RATING) is a rating of purchase intent on a five-point scale. The three attributes listed above will be used as predictor variables in the model. We will also include the two demographic variables as covariates, in a second model.

- **SEX** (1 = Male; 2 = Female)
- **AGE** (1 = 16-24; 2 = 25-39; 3 = 40+).

### 8.2 Using a GUI example to setup the Syntax model

- Navigate as in Figure 8-2 to open the Example GUI model named ‘Tutorial S1: LC Regression’.
As shown in Figure 8-3, the Outline pane contains the name of the data file along with the previously saved model(s). The Contents Pane (currently empty) shows the contents of selected model output.

Double click to open the Analysis Dialog Box for this model.

For reasons explained earlier in “The Goal” section, we will now change the scale type from Ordinal (“Ord-Fixed”) to ‘Continuous’ to specify a (3-class) linear regression model.

- Right click on the dependent variable ‘rating’ to view the available scale types
- Select ‘Continuous’ (see Figure 8-4)
Select ‘Estimate’ to estimate this GUI model.

A Warning Message appears alerting you to the fact that the dependent variable contains fewer than 20 values.

Select ‘OK’ to estimate the linear regression model anyway, despite the small number of values for the dependent variable.

Click on ‘Parameters’ to view the Parameters output in the Contents Pane (see Figure 8-5).
Right click to retrieve the popup menu (shown above) and select ‘Std Errs & Z’ to display these statistics.

Notice that the regression coefficient for FASHION is not significant for class 3 (as highlighted in Figure 8-6, \(|Z| < 2\)).
Double click on the model name ‘3-class Regression Model’ or the new model named ‘Model2’ to re-open the Analysis Dialog Box (the new model is a copy of the last estimated model).

Click on ‘Model’ to open the Model Tab.

Right click on the ‘3’ associated with the FASHION coefficient for latent class 3, and select ‘No effect’ from the Pop-up menu to restrict this effect to 0 (see Figure 8-7).

Click Estimate to estimate this restricted model.

Figure 8-7: GUI Model Tab
Click on the model name (‘Model2’) to select it, click again to enter the edit model, and type ‘3-class restricted model’ to rename it (see Figure 8-8).

To generate the syntax specifications for all models listed in the Outline Pane for the Conjoint.sav’ file:

- Select the file name ‘Conjoint.sav’
- From the Model Menu, select ‘Generate syntax’

A separate (new) syntax tree appears above the GUI tree as shown in Figure 8-9, similar to the appearance when you open a different dataset in the GUI. Note that the GUI model names are preserved in the syntax. The first of these models, the unrestricted model, is highlighted and the syntax specifications for it appear in Contents Pane. It can be edited.
At the bottom of the syntax three equations appear, each ending with a ‘;’. The special keyword ‘1’ denotes the intercepts.

- The first equation corresponds to the Model for the Classes, which contains only the intercepts. These intercepts are logit parameters, which yield the class size probabilities.

- The second is the linear regression equation with terms separated by ‘+’, coefficients to be estimated for each such term. The skeletal structure of this model is expressed by ‘rating = 1 + fashion + quality + price’, with conditional effects ‘( | Class)’ specified for each effect. ‘Conditional effects’ means that separate coefficients are estimated for each latent class.

- The third equation specifies that separate error variances are estimated for each class.

### 8.3 Estimating a Syntax model

To estimate this model:

- From the Model menu choose ‘Estimate’

or, you may also select the on the toolbar
Upon completion of the estimation, the log-likelihood (LL) appears in the Outline Pane to the right of the model name (see Figure 8-10), and the syntax and output listings appear as separate entries in the expanded syntax tree. The model output listings in this expanded syntax tree appear somewhat differently than in the corresponding version of the expanded GUI tree (recall Figure 8-5):

- The syntax statements appear as an additional entry named ‘syntax’
- The ProbMeans output is named ‘ProbMeans-Posterior’ to distinguish it from other versions of the ProbMeans that are available in the syntax.
- The Estimated Values output is named ‘EstimatedValues-Regression’ to distinguish it from other versions of the Estimated Values that are available in the syntax.
- The Bivariate Residuals output appears as an additional entry because it is available (by default) in the syntax but not available at all in the GUI for regression models.

The equivalence of the unrestricted models estimated from the syntax and GUI can be confirmed by comparing the LL values and verifying that they are equal (LL = -4673.173).

Select ‘Parameters’ to view the Parameters output in the Contents Pane.

Notice that the Parameters output is formatted differently than the GUI. Among the differences, a p-value column is present in the syntax output, showing that the Z-value of -1.3 (highlighted in Figure 8-10) is not significant at the .05 level (p=.18).

Figure 8-10: Syntax Parameters Model Output

Unlike the results from a model estimated in the GUI, the latent classes in an unrestricted syntax model will not necessarily be ordered from high to low. Figure 8-11 (below) shows that
in this particular run, latent class 2 is the largest class, while class 1 is the second largest. If we estimated the model again, the ordering may be different.

To view the classes sizes:

- Select ‘Profile’ in the syntax tree.

**Figure 8-11: Profile Output with Class Sizes Highlighted for Unrestricted Model**

8.4 **Restricting certain effects to be zero or class independent**

The syntax allows substantial flexibility in placing restrictions on the model parameters. To view the specification generated automatically from the restricted GUI model:

- Select the model name ‘3-class restricted model’

The syntax appears in the Contents Pane.

**Figure 8-12: Syntax Equations for 3-class Restricted Model**

Notice that an additional equation appears that embodies the parameter restriction. $B1(3) = 0$ states that the parameter $B1$ for latent class 3 is set to 0. The $B1$ coefficients are defined in the regression equation by adding ‘(B1)’ as part of the conditional effect term for the predictor FASHION. This defines 3 $B1$ parameters, one for each class. These parameters are referenced as $B1[1]$, $B1[2]$ and $B1[3]$. By restricting $B1[3] = 0$, this serves to define latent class 3 as the one for which the $B1$ effect is zero.
To estimate this model,

- select \[ \text{button} \] on the toolbar
- Select the data file name to see how the models compare.

**Figure 8-13: Model Summary Display**

The restricted model has one fewer parameter (Npar = 16 vs. 17 for the unrestricted model), is preferred according to the BIC (9444 vs. 9448) and both models have an \( R^2 = .60 \).

- Select ‘Profile’ to view the class sizes:

**Figure 8-14: Profile Output for Restricted Syntax Model**

The Figure above shows that the largest class is now the first class. If we re-estimated this restricted model again, classes 1 and 2 may be reversed, but class 3 will remain as class 3 because of the restriction. Comparing the Parameter output between the 2 models, you will find that the parameter estimates did not change much, if you keep in mind that the ordering of the classes may have changed.

- Select ‘EstimatedValues-Regression’ to view the expected ratings.
Since the ratings for class 3 are not affected by FASHION, we see that the expected rating for this class is the same when evaluating Traditional vs. Modern shoes of the same PRICE and QUALITY. This is not the case for classes 1 or 2.

### 8.5 Inclusion of covariates in model

Next, we will expand the model for the classes to include the covariates. Main effects for the covariates could be specified in the GUI, and a syntax model could have been generated as before. But here, we will use the syntax to specify both main effects and interaction effects for the covariates.

- Select ‘Model 3’ to view a copy of the syntax for the 3-class restricted model in the Syntax Editor.
- Scroll down to the Independent Variables Section.
- Type “age, sex” to include the covariates AGE and SEX as additional independent variables (see Figure 8-16B). Alternatively, if you right click in the Contents Pane, you can retrieve a list of data file variables from which you can select the covariates and the variable names will be copied to the syntax (see Figure 8-16A)
- Type “+ age + sex + age sex” to include the main effects and interaction effect of the covariates in the model for classes.
- Select on the toolbar to estimate this model.
Select ‘Model 4’ to view a copy of the syntax for the last model estimated.

Scroll down to the Model for classes.

Remove the interaction effect.

Select on the toolbar to estimate this model.

Select the data file to compare all models.

According to the BIC, both Models 3 and 4 are preferred over the models without these covariates. Model 4 provides the best fit suggesting that the interaction effect is not significant.

Select ‘Parameters’ for Model 4 to display the Parameters output for this model.

Note that the covariates AGE and SEX are both significant in the Model4.
Select ‘Bivariate Residuals’ for Model4 to display these statistics

Note that the BVRs for both covariates are small, suggesting that the assumption that the relationship between the covariates and dependent variable is explained by the latent classes. The BVRs are not available in the GUI for regression models. If one or more of these BVRs were large, the covariate(s) with the large BVRs could be included also as a predictor in the regression model.
Figure 8-19: Bivariate Residuals – Regression Display for Model with Covariates

<table>
<thead>
<tr>
<th>Dependent</th>
<th>Independent</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fashion</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>quality</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>price</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>age</td>
<td>1.8810</td>
</tr>
<tr>
<td></td>
<td>sex</td>
<td>0.0121</td>
</tr>
</tbody>
</table>
References


Bakk, Zs. and Vermunt, J.K. (2013). Robustness of stepwise approaches to latent class modeling with continuous distal outcomes. *Under review*.


## Index

### A

alpha · 17, 23, 27  
append · 18, 24, 27, 29, 45, 46, 84, 85, 92

### B

bootstrap · 17, 19, 26, 27, 29, 30, 31, 81, 86, 87, 88, 92  
bvr · 24, 32  
bvrlongitudinal · 24  
bVRTwoLevel · 24

### C

case · 10, 15, 28, 29, 31, 32, 33, 34, 36, 40, 48, 54, 58, 61, 63, 65, 73, 76, 77, 80, 82, 87, 91, 92, 95, 105  
caseid · 32, 33, 62, 68, 70, 71, 73, 84  
caseweight · 33, 81, 83, 84  
categorical · 6, 16, 23, 29, 30, 32, 49, 53, 67, 68, 88, 93, 109  
censored · 33, 34, 49  
classification · 24, 27, 28, 29, 30, 31, 32, 35, 37, 64, 69, 70, 73, 80, 93  
coding · 19, 20, 30, 33, 35, 36, 37, 42, 45, 46, 47, 53, 63, 65, 72, 83  
continuous · 6, 15, 16, 20, 27, 28, 29, 31, 33, 34, 35, 36, 37, 39, 40, 41, 46, 47, 49, 52, 53, 57, 58, 62, 63, 64, 66, 67, 68, 70, 71, 72, 73, 79, 83, 94, 109  
cooksd · 24  
cumlogit · 33, 34, 67, 69, 84

### D

dependent · 6, 14, 15, 16, 18, 19, 20, 26, 28, 29, 30, 31, 33, 34, 35, 37, 39, 40, 41, 42, 45, 46, 47, 49, 52, 58, 61, 62, 64, 67, 68, 69, 70, 71, 73, 77, 79, 81, 83, 84, 85, 88, 90, 94, 95, 96, 97, 107  
distribution · 36, 68  
dynamic · 15, 28, 29, 36, 45, 62, 71, 72, 73

### E

effect · 14, 20, 24, 30, 40, 44, 49, 51, 53, 54, 56, 58, 61, 62, 64, 71, 77, 82, 85, 99, 101, 103, 105, 106  
e · 24, 31  
eiterations · 23, 27, 32, 77  
etolerance · 23, 27, 32  
equations · 14, 22, 39, 40, 41, 45, 47, 48, 49, 51, 52, 55, 57, 58, 60, 63, 64, 67, 68, 69, 70, 71, 72, 73, 77, 78, 82, 83, 85, 101  
estimatedvalues · 24, 29, 30, 32, 69, 70, 71  
excludeall · 23  
exposure · 20, 33, 34, 35

### F

fast · 24, 30, 87  
first · 6, 20, 22, 24, 28, 30, 32, 33, 35, 36, 37, 42, 43, 44, 45, 52, 53, 57, 65, 69, 74, 77, 93, 100, 101, 104  
forward · 24, 31  
frequencies · 18, 24, 30, 93

### G

group · 14, 15, 19, 20, 28, 32, 33, 36, 37, 54, 64, 65, 66, 82, 83, 84  
groupid · 32, 33, 65, 71  
groupweight · 33, 81

### H

hblike · 24, 30, 31, 91  
holdout · 14, 17, 18, 38, 91

### I

id · 15, 25, 28, 31, 38, 68, 70, 71, 73, 82  
identification · 2, 6, 18, 24, 29, 30, 32, 41, 57, 88  
ignoreknownclass · 24, 30  
imputation · 14, 17, 24, 26, 31, 39, 76, 81, 82, 84, 109  
imputationsid · 17, 31, 38, 39, 82  
inactive · 35  
includeall · 23, 26, 32, 63  
includependent · 23  
independent · 15, 16, 19, 20, 26, 28, 29, 31, 35, 37, 39, 41, 44, 45, 47, 49, 52, 53, 54, 56, 59, 61, 62, 64, 67, 68, 69, 70, 71, 73, 83, 85, 93, 103, 105  
individualcoefficients · 24, 31  
infile · 21, 22, 80  
int · 6, 14, 46, 47, 72, 94  
intercept · 20, 39, 40, 41, 43, 46, 47, 51, 57, 58, 60, 62, 64, 72, 77  
iterationdetails · 23, 27, 32  
iterations · 17, 18, 23, 25, 27, 32, 48, 77, 80, 81
**J**

jackknife · 19, 24, 26, 29, 30, 87, 88

**K**

keep · 25, 31, 32, 104
knownclass · 36, 37, 67, 73, 74

**L**

L2 · 23, 26, 86, 92
loglog1 · 33, 34, 67
loglog2 · 33, 34, 67
longitudinal · 19, 24, 28, 30, 38, 109

**M**

markov · 36
mis · 46, 47
missingvaluedummy · 20, 46
mode · 7, 10, 13

**N**

n · 10, 16, 17, 18, 29, 38, 81, 82, 91
nodes · 20, 24, 36
nominal · 15, 16, 18, 20, 30, 33, 34, 35, 36, 37, 39, 41, 42, 44, 45, 46, 47, 49, 51, 53, 54, 55, 57, 58, 60, 62, 64, 66, 67, 68, 69, 70, 71, 72, 73, 74, 76, 80, 83, 84, 88, 90
none · 24, 27, 28, 70, 71
npbootstrap · 24, 30
nriterations · 23, 27, 32, 77, 81
numeric · 35, 38, 45, 49, 53, 73

**O**

options · 6, 8, 11, 14, 16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31, 32, 35, 37, 38, 41, 46, 48, 50, 63, 67, 69, 70, 71, 72, 73, 74, 76, 77, 81, 82, 83, 84, 85, 86, 87, 88, 91, 92, 93
ordinal · 6, 15, 16, 18, 20, 30, 33, 34, 35, 36, 37, 41, 46, 47, 49, 51, 54, 55, 58, 67, 68, 69, 70, 71, 73, 80, 84, 94
outfile · 24, 31, 32, 48, 50, 80, 81, 82, 84
Output · 7, 8, 9, 10, 12, 18, 19, 23, 28, 29, 30, 42, 76, 98, 102, 103, 104, 105, 107
overdispersed · 33, 34, 35, 39, 40, 49, 52

**P**

parameters · 6, 14, 16, 17, 18, 19, 20, 22, 24, 27, 29, 30, 31, 32, 39, 42, 43, 44, 45, 46, 47, 48, 52, 53, 54, 63, 68, 69, 70, 71, 72, 76, 77, 78, 79, 81, 82, 84, 85, 86, 87, 88, 90, 93, 101, 103
pbootstrap · 24, 30, 86
poisson · 23, 32, 33, 35
populationsize · 37
posterior · 19, 24, 28, 29, 30, 31, 35, 36, 37, 69, 70, 71, 73, 74, 91, 92
power · 6, 17, 19, 23, 26, 27, 29, 48, 76, 82, 83, 84, 85, 86, 87, 92, 93
prediction · 18, 24, 28, 30, 31, 38, 80, 91
predictionstatistics · 24
probit · 16, 33, 34, 67, 84
probmeans · 24, 29, 30, 32
profile · 24, 29, 30, 32, 70
psuid · 18, 37

**R**

rank · 18, 29, 33, 35, 88
regression · 6, 14, 15, 16, 22, 24, 28, 31, 32, 33, 34, 39, 40, 41, 44, 45, 49, 51, 53, 57, 58, 60, 61, 62, 65, 66, 67, 68, 69, 71, 72, 94, 96, 97, 98, 101, 102, 103, 107
replicates · 16, 17, 23, 27, 30, 84, 85, 87, 88
replicationweight · 33, 81
rescale · 37
robust · 24, 30, 87
rsweights · 37

**S**

samplesizeBIC · 24
samplingweight · 37
score · 6, 19, 29, 33, 35, 37, 44, 48, 80, 87, 93
scores · 20, 36, 37, 47
scoretest · 24, 29
scoring · 11, 14, 16, 19, 20, 22, 29, 70, 76, 80, 93
seed · 23, 24, 25, 27, 31, 32
select · 7, 10, 11, 20, 32, 38, 50, 80, 98, 99, 100, 101, 104, 105
seqlogit1 · 33, 34, 67
seqlogit2 · 33, 34, 67
sets · 6, 12, 14, 16, 17, 19, 20, 23, 25, 26, 27, 29, 30, 31, 32, 37, 39, 51, 61, 76, 79, 81, 82, 83, 84, 85, 88
simulation · 6, 14, 16, 17, 19, 24, 26, 27, 29, 31, 34, 48, 76, 77, 82, 84, 85, 86, 92
simulationid · 16, 31, 38, 82, 84
standard · 6, 7, 11, 16, 17, 18, 19, 24, 25, 28, 29, 30, 31, 37, 39, 55, 68, 69, 81, 82, 84, 85, 86, 87, 88, 93, 109
standarderrors · 24, 32, 69, 70, 71, 86
stratumID · 37
timeid · 32
timeinterval · 37, 63, 72, 73
timelabel · 37, 38
title · 22
tolerance · 23, 27, 32
truncated · 33, 34, 35, 49

validationid · 18, 38, 91
validationLL · 24, 38, 91
variances · 14, 15, 23, 28, 32, 39, 48, 52, 59, 78, 82, 101
vonmises · 33, 34

waldpower · 24, 29