LG-SYNTAX™ USER'S GUIDE: MANUAL FOR LATENTGOLD® SYNTAX MODULE VERSION 6.1[1]

JEROEN K. VERMUNT AND JAY MAGIDSON

(January 2025)

[1] This document should be cited as: Vermunt, J.K., and Magidson, J. (2025). LG-Syntax User's Guide: Manual for LatentGOLD Syntax Module Version 6.1. Tilburg, The Netherlands: Statistical Innovations Europe.

For more information about Statistical Innovations please visit our website at

http://www.statisticalinnovations.com or contact us by e-mail: support@statisticalinnovations.com

LatentGOLD, LG-Syntax, and LG-Equations are trademarks of Statistical Innovations.

Windows is a trademark of Microsoft Corporation.

SPSS is a trademark of IBM Corporation.

Stata is a trademark of StataCorp LLC.

Other product names mentioned herein are used for identification purposes only and may be trademarks of their respective companies.

LG-Syntax™ User's Guide: Manual for LatentGOLD® Syntax Module Version 6.1.

Copyright © 2008, 2013, 2016, 2021, 2025 by Statistical Innovations.

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 Europe.

CONTENTS

Contents 3

1 Introduction 7

2 The LG-Syntax User Interface 8

2.1 Setting up a Syntax session and saving Syntax models 8

2.2 Using the *Save Syntax* option 11

2.3 Saving output 12

2.4 Using the Examples menu 12

2.5 Operating the program in batch mode 13

3 Overview of the Extended Features of Syntax 15

3.1 More flexible modeling and parameter restrictions using LG-Equations™ 15

3.2 Additional models compared to the Cluster, DFactor, Regression, Step3, Markov, and Choice 15

3.3 Monte Carlo simulation options 17

3.4 Multiple imputation 18

3.5 Validation and holdout options 18

3.6 Output and saving options 18

3.7 Scoring new cases 20

3.8 Other options 20

4 The Syntax Language 22

4.1 Options section 23

4.1.1 Maxthreads 26

4.1.2 Algorithm 26

4.1.3 Startvalues 27

4.1.4 Bayes 28

4.1.5 Missing 28

4.1.6 Montecarlo 28

4.1.7 Quadrature 30

4.1.8 Step-three options 30

4.1.9 LC tree options 31

4.1.10 Factor rotation options 31

4.1.11 Output options 32

4.1.12 Outfile options 35

4.1.13 An example 37

4.2 Variables section 37
4.2.1 ID variables 37
4.2.2 Frequency weights 38
4.2.3 Dependent variables 38
4.2.4 Independent variables 41
4.2.5 Latent variable information 42
4.2.6 Survey options 44
4.2.7 Options for Markov models 44
4.2.8 Other options 45
4.3 Equations section 46
4.3.1 Regression equations 46
4.3.2 Variances 47
4.3.3 Covariances and associations 47
4.3.4 Log-linear scale factor equations 47
4.3.5 Using lists of variables 48
4.3.6 Parameter restrictions 48
4.3.7 Starting values 51
4.3.8 Restrictions or starting values for a list of parameters 51
4.3.9 Features for Markov models 52
4.3.10 Special parameter coding using the tilde character (~) 52
4.3.11 Starting values and saved parameters between '{ }' or in separate text file 54
4.3.12 Reading fixed values as well as their variance-covariance matrix from a text file 56
4.3.13 Unallowed model specifications 56
4.3.14 User-defined expressions 57
4.4 Syntax specific for choice models 58
4.4.1 Choice File FormaTs 58
4.4.2 Choice options 59
4.4.3 Choice variables 59
4.4.4 Choice equations 61
4.5 Selecting a keyword, variable, file, or model 61
5 Model Examples 62
5.1 Cluster models 62
5.1.1 Unrestricted cluster model 62
5.1.2 Restricted cluster model 63

5.1.3 Cluster model with local dependencies 64
5.1.4 Cluster model with covariates 64
5.1.5 Multiple-group cluster model 64
5.2 Discrete-factor models and other models with multiple discrete latent variables 65
5.2.1 Single DFactor model 65
5.2.2 Exploratory two-DFactor model 66
5.2.3 Confirmatory two-DFactor model 66
5.2.4 Confirmatory two-DFactor with a path model for the DFactors 66
5.2.5 Confirmatory two-DFactor model with a path model containing independent variables 67
5.3 Factor analysis and IRT models 67
5.3.1 Single factor or IRT model 67
5.3.2 Confirmatory 2-factor or 2-dimensional IRT model 68
5.3.3 Factor analysis or IRT models with covariates 68
5.3.4 Latent growth models 68
5.3.5 Latent growth models as models for a repeated univariate response 70
5.4 Latent class regression models 71
5.4.1 Simple LC regression model 71
5.4.2 LC regression model for two or more dependent variables 71
5.4.3 LC regression model with a random intercept 72
5.5 Latent Markov models 72
5.5.1 Time-homogeneous latent Markov model 73
5.5.2 Latent Markov model with time-heterogeneous transitions 73
5.5.3 Latent Markov model with a grouping variable 74
5.5.4 Latent Markov model with a covariate 74
5.5.5 Mixture (or mixed) latent Markov model 74
5.6 Multilevel LC models 75
5.6.1 Simple multilevel LC model 75
5.6.2 Multilevel LC Model with Covariates 75
6 Details on Various Syntax Modeling Options 76
6.1 Alternative regression models for dichotomous and ordinal dependent variables 76
6.2 Additional regression models for continuous dependent variables 77
6.3 Bias-adjusted step-three modeling 77
6.4 Regression models with a cell weight vector ('~wei' option) 80
6.5 Modeling the effect of independent variables using b-splines 81

6.6 Continuous-time Markov models 82
6.7 Knownclass option 83
6.8 Log-linear scale factor models for dependent variables 84
6.9 Exploratory factor analysis with rotation to simple-structure and agreement 85
6.10 Using a dynamic latent variable to define an additional level in a multilevel LC model 86
6.11 Syntax extension for choice models 86
6.11.1 Example 1: A basic choice model 87
6.11.2 Example 2: Data structure for choice plus other dependent variables 88
6.11.3 Example 3: Data fusion example with a scale model and a continuous factor 89
7 Details on Various Syntax Tools and Output Options 90
7.1 Internal parameter structure 90
7.2 Obtaining additional output and scoring external files 94
7.3 Multiple imputation of missing values 95
7.4 Monte Carlo simulation and power calculation features 96
7.5 Bootstrap, Vuong, and asymptotic likelihood-ratio tests 99
7.6 User-defined Wald tests 101
7.7 User-defined expressions 102
7.8 Power computation for Wald tests 103
7.9 Asymptotic power computation for likelihood-ratio tests 103
7.10 Score tests and EPCs 104
7.11 Alternative standard error estimators 104
7.12 Identification checking 105
7.13 Validation and hold-out options 106
7.14 Options to write output to text files 107
8 Getting Started with LG-Syntax 110
8.1 The application and the data 110
8.2 Using a GUI example to setup the Syntax model 111
8.3 Estimating a Syntax model 116
8.4 Restricting certain effects to be zero or class independent 117
8.5 Inclusion of covariates in model 119
References 123

1 INTRODUCTION

The main goal in the development of LG-Syntax™ was to offer LatentGOLD 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, perform power analyses, use different types of variance estimators, use different types of link function for ordinal and continuous variables, 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™ system, which is part of the Advanced/Syntax add-on to LatentGOLD release 6.1. LG-Syntax™ extends the capabilities of both LatentGOLD Basic as well as the LG-Choice add-on module in several important ways. The accompanying 'Upgrade Manuals for LatentGOLD 5.0, 5.1, and 6.0/6.1' provide all details regarding the changes between versions 4.5 and 5.0, versions 5.0 and 5.1, and versions 5.1 and 6.0/6.1, respectively. 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 the LG-Syntax module, and compares it with the LatentGOLD Basic, Advanced, and Choice modules.

Section 4 spells out the full Syntax Language. In this section, the overall structure of a LatentGOLD Syntax ('.lgs') file is described with detailed documentation for each of the three 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, bias-adjusted step-three modeling, models using cell weights, modeling of effects of independent variables using B-splines, continuous-time latent Markov models, models using known-class information, log-linear scale factor models for categorical dependent variables, multi-group and mixture exploratory factor analysis models with factor rotation, and basic and advanced choice models.

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 scoring other data files, multiple imputation, Monte Carlo simulation, specification of user-defined Wald tests, obtaining functions of model parameters, power computation for Wald and likelihood-ratio tests, obtaining Score tests for constrained parameters, identification checking, requesting alternative complex sampling standard errors, holding out replications or cases for validation, and writing 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 in the Help / Examples 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.

Syntax models can be specified, structured, and estimated using the LatentGOLD Windows Menu system in conjunction with the LG 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 lg61.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, saving output, the use of examples from the help menu, and running LatentGOLD 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 LatentGOLD using the 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 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 LatentGOLD 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 Syntax editor. For example, Figure 2-1 shows a 'Syntax Tree' with the model named '3-class' highlighted, after being generated automatically from the GUI model with the same name. The GUI model appears directly below the 'Syntax Tree' in Figure 2-1. Since the Syntax model '3-class' is highlighted in Figure 2-1, the associated Syntax statements are shown in the Output Pane, organized into three primary sections – Options, Variables and Equations.

The ‘Generate Syntax’ command can also be executed from the ‘Model’ menu immediately, without setting up any initial model at all, but then only the Options Section appears (with the default options) and the statements in the Variables and Equations Sections would need to be created by hand in the Syntax editor.

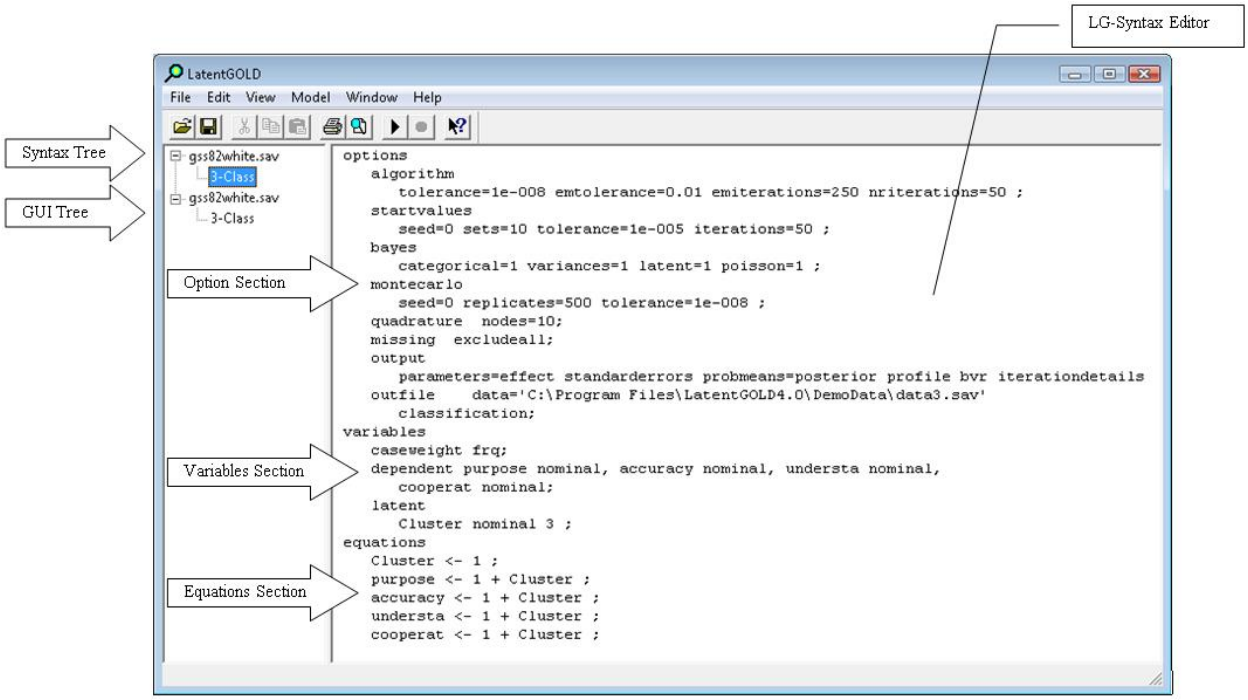
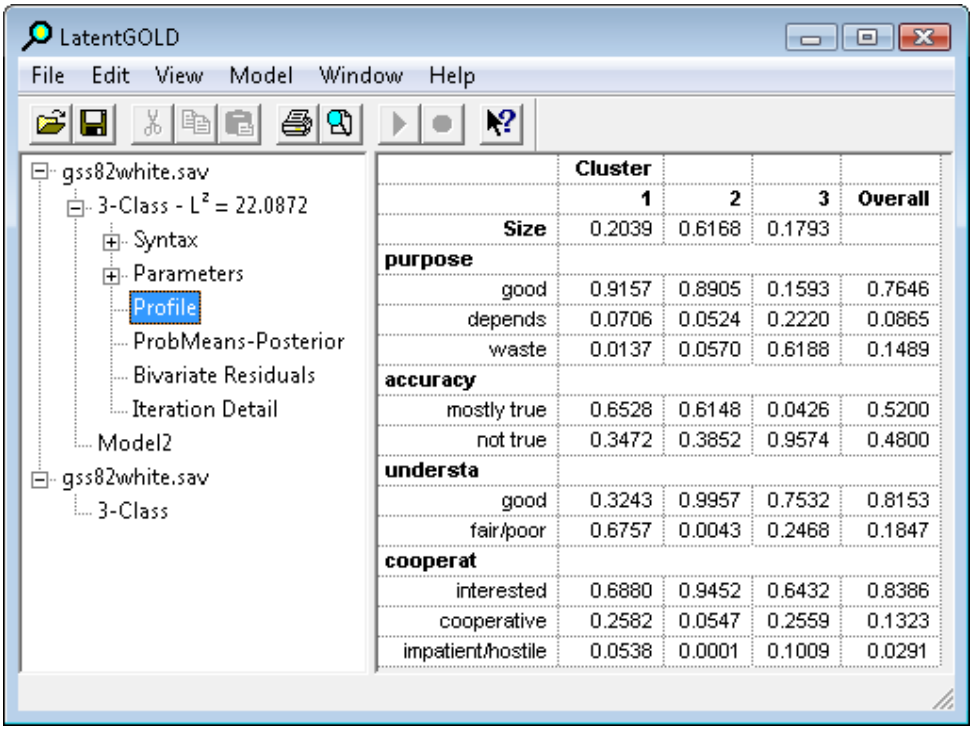


Figure 2-1: Example of a Syntax Model Generated from a GUI Model

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 associated with a new model (named ‘Model2’ in Figure 2-2). Simply modify the Syntax statements as desired, and then execute the ‘Estimate’ command.

As the GUI modules, LG Syntax can be used to estimate a range of models with different numbers of classes, say models with 1 to 4 classes. While in GUI models this is achieved by specifying ‘1-4’ in the Clusters/Classes/States section of the GUI dialogue box, in Syntax models this requires specifying ‘1:4’ for the number of classes (thus using a colon instead of a hyphen). This not only simplifies the estimation of models with different numbers of classes, it also allows obtaining Vuong-Lo-Mendell-Rubin and bootstrap likelihood-ratio tests comparing models with one class difference.



	Cluster			
	1	2	3	Overall
Size	0.2039	0.6168	0.1793	
purpose				
good	0.9157	0.8905	0.1593	0.7646
depends	0.0706	0.0524	0.2220	0.0865
waste	0.0137	0.0570	0.6188	0.1489
accuracy				
mostly true	0.6528	0.6148	0.0426	0.5200
not true	0.3472	0.3852	0.9574	0.4800
understa				
good	0.3243	0.9957	0.7532	0.8153
fair/poor	0.6757	0.0043	0.2468	0.1847
cooperat				
interested	0.6880	0.9452	0.6432	0.8386
cooperative	0.2582	0.0547	0.2559	0.1323
impatient/hostile	0.0538	0.0001	0.1009	0.0291

Figure 2-2: Selecting the Profile output section makes it visible in the Output Pane

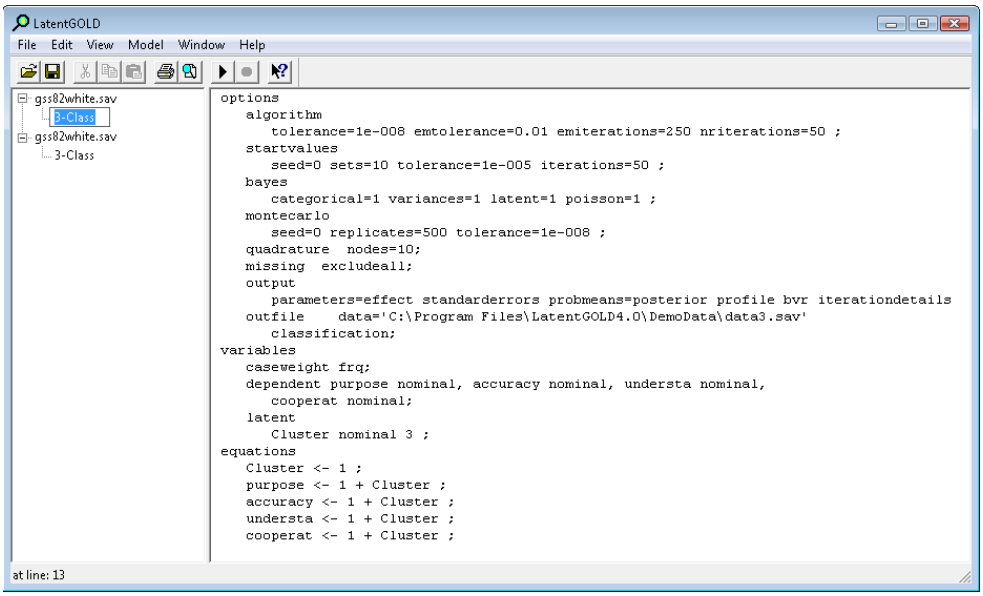


Figure 2-3: Editing a Model Name

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, the model is estimated.

After a Syntax model has been estimated, the Syntax statements can be viewed in the Output Pane by clicking to highlight the 'Syntax' tree entry in the Outline Pane. Since these statements now serve to document the model just estimated, they can no longer 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.

2.2 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' (LatentGOLD File) format. The 'Save Syntax' option saves a current model setup in a 'lgs' (LatentGOLD 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 sections 3.7 and 7.2 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 three '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).

3. Syntax with Parameters and Variable Definitions: model setup + parameter estimates + detailed data information.

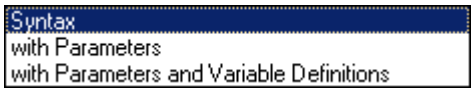


Figure 2-4: The drop-down menu in the 'Save as Type' box lists these 'Save as type' options:

2.3 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'.

2.4 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.

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)'

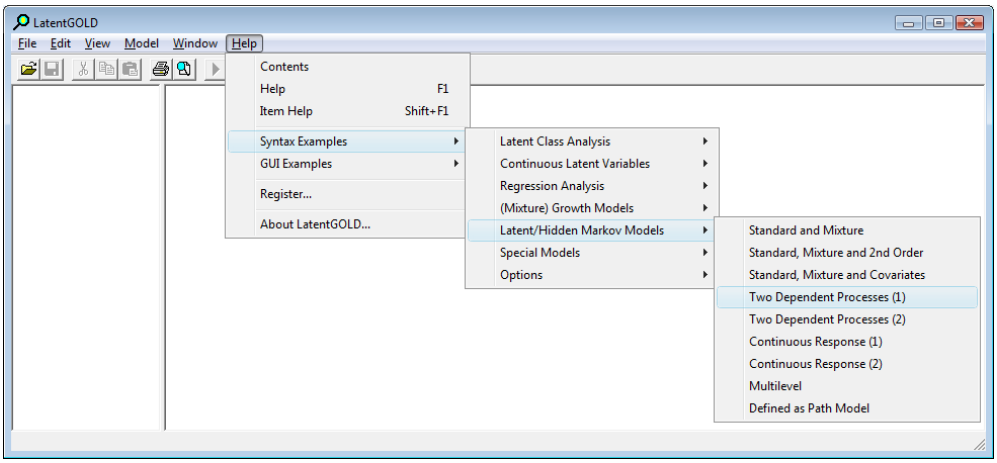


Figure 2-5: The Examples Menu

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 8: 'Getting Started with LG-Syntax' for further instruction regarding the use of the Examples menu.

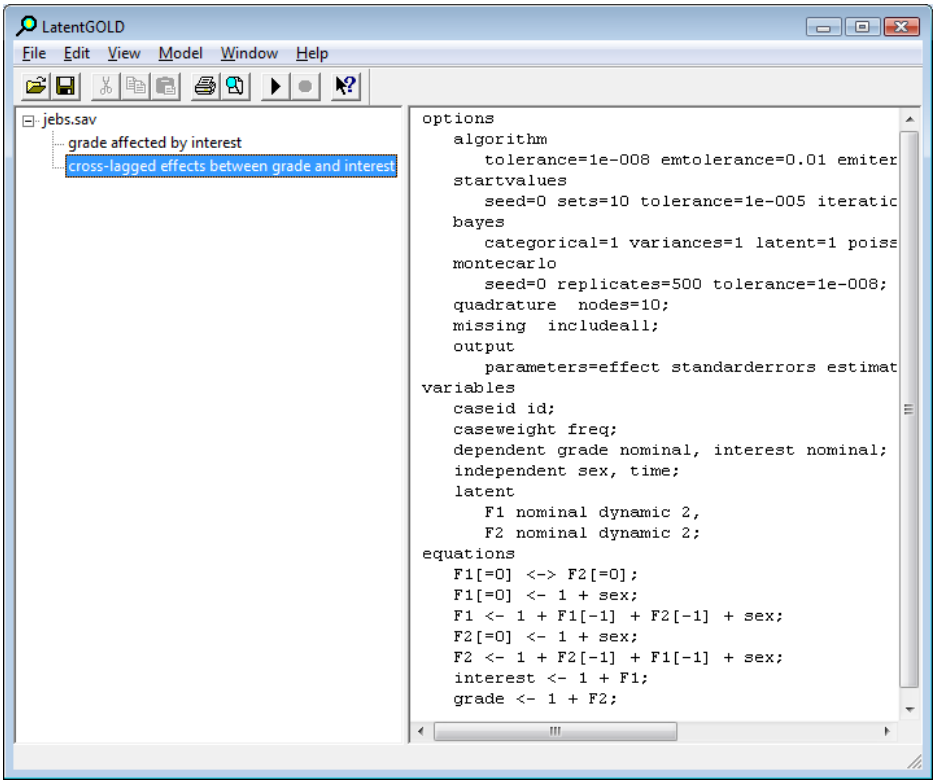


Figure 2-6: The Syntax Specifications for the 'cross-lagged effects between grade and interest' example

2.5 OPERATING THE PROGRAM IN BATCH MODE

LatentGOLD can run models in 'batch mode' via the Windows command prompt. The installation program provides a Start Menu item labeled 'LatentGOLD Command line' which starts up a Windows command prompt that allows the command 'lg61' to be recognized (via the insertion of the application directory into the PATH environment variable).

Models are run by issuing the command:

lg61 <list of model files> /b /o <outputfile> /h /l /r <number> /fmt [f|g|e]<number>

where <list of model files> contains the name(s) of the file(s) (including the path) with the LatentGOLD model definitions. Both GUI (with a .lgf suffix) and Syntax (.lgs) models can be run in this manner. From version 6.1 on LG-Syntax allows specifying multiple model files (which are run in the specified order).

The command line option '/b' is required for running in batch model. Excluding this switch will simply open the model file in the LatentGOLD GUI (Graphical User Interface) rather than starting a batch run.

The '/o' switch followed by a file name can be used to set the name of the output file. By default, it is derived from the model file name: 'modelfile.lst'.

The command line option '/h' will cause the output listing to be in html style output, instead of a plain text file.

The '/l' (lowercase L) switch indicates that the log (or iteration details) information should be shown on the screen, which may be useful if one wishes to see how the model estimation is progressing.

The '/fmt' switch can be used to specify the formatting of the numbers in the output file. Here, f is fixed format, g is general format, and e is exponential/scientific format. These are immediately followed by <number>, which is an integer specifying the number of decimals (with f) or significant digits (with g or e).

The specified lgs file(s) can be run multiple times, which is indicated with the command line switch '/r <number>', where <number> refers to the number of replicates. This option is mainly of interest for simulation studies, where the first lgs file will typically simulate a data set, and the next ones will estimate the models for interest with the simulated data set (or with a newly created data set using the LG-Syntax outfile option). When using this option, LatentGOLD will be started only once, which saves the time that would otherwise be needed to load the program.

It you be noted that the lg61.exe file may also be executed from within other environments, such as R. The latter may be useful, for example, when performing a simulation study where LatentGOLD does the actual model estimation part.

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 LatentGOLD GUI modules. These include the following:

- More flexible modeling and parameter restrictions by specifying intuitive LG-Equations™
- Additional models compared to the Cluster, DFactor, Regression, Step3, Markov, and Choice
- Monte Carlo simulation options
- Multiple imputation options
- N-fold validation and other holdout options
- Additional output and saving options
- Options to use saved parameters (e.g., for scoring)
- 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 parameter may be specified in a way that they 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 been 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 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.
- LG-Syntax implements various options for exploratory factor analysis with continuous dependent variables. The most important one is the possibility to perform factor rotation, to simple structure and to agreement between groups or latent classes.
- LG-Syntax allows specifying (mixture) SEM models for latent factors measured with continuous indicators or saved factor scores (see information below on three-step models with factor scores).
- 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 Markov GUI submodule, which is part of the Advanced/Syntax module. LG-Syntax 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). Moreover, it is possible to specify continuous-time (latent) Markov models, which are models that accommodate measurements which are not equally spaced in time. The dynamic latent variable option can also be used to define multilevel LC models with arbitrary numbers of levels.
- LatentGOLD Basic includes the Step3 GUI submodule for bias adjusted step-three modeling (and scoring). LG-Syntax provides various more extended step-three modeling possibilities, such a step-three models with both covariates and dependent variables, step-three models with multiple latent variables, step-three latent Markov models, and step-three multilevel LC models.
- LG-Syntax implements three-step analysis for continuous latent variables. Saved factor scores (posterior means) with standard error estimates (posterior standard deviations) are used as input. This can be combined with the SEM features (Vermunt, 2025).
- For ordinal dependent variables, in additional to the adjacent-category logit model, in LG-Syntax one can specify models based on cumulative responses probabilities -- cumulative logit, probit, log-log, loglinear, and linear models -- as well as models for continuation-ratio or sequential logits. These specifications can also be used with dichotomous responses.
- LG-Syntax contains additional 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. It also allows modeling interval censored normally distributed responses.

- The LG Syntax can be used for modeling multivariate counts using multinomial or multivariate Poisson distributions (including version with truncation and overdispersion), and for modeling multivariate responses which sum to 1 using a Dirichlet distribution.
- The LG Syntax implements survival models with parametric, semi-parametric, or non-parametric specifications of the time dependence. Depending on the type of model, it is available as hazard rate model, accelerated failure time model, or both.
- In LG Syntax it is the possibility to specify a log-linear scale factor model for categorical response variables (nominal, all ordinal types, all choice types) and for continuous and count responses. Since for categorical responses the scale factor is inversely related to the response uncertainty, this option makes it possible to model heterogeneity in response (un)certainty. The log-linear scale model may contain independent variables, as well as discrete and continuous latent variables as predictors.

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 as well as 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 deviations across simulation replicates. It is also possible to write the results for all simulation replicates to a compact 'csv'-type output file.
- Another simulation option is provided by the 'MCstudy=<model name>' option. This, in fact, fully automates the two steps described the previous option 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.
- The command line options allow running multiple lgs files multiple times, which makes it straightforward to perform a simulation study. The first lgs file in the list of lgs files can be used to simulate a single data set, while the remaining lgs files will estimate the models of interest with the simulated data set. This is then repeated as many times as needed with the '/r' switch. The output from the multiple replicates can be stored to one or more compact 'csv'-type output files using the output option 'append=<filename'.
- 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. Both types of bootstrap procedures can 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, 2016a/b), 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, but also an experimental Gibbs sampling procedure has been implemented. 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 LatentGOLD. Any type of LatentGOLD 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 procedure 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 statistics, 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 LatentGOLD.

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 defined 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 prediction statistics are also computed using the other set of the replications/responses for the same cases. Holdout replications are defined 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.

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 obtain *marginal effects*, which are partial effects on probabilities or expected values (Long, 1997; Long and Freese, 2014). LG Syntax implements average marginal effects (as well as their standard errors, z values, and p values) for all types of regression parameters, including the logit coefficients for covariate effects on latent classes or transition probabilities. For nominal predictors, the program will also report the average marginal mean for each category.

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. It is also possible 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, this statistic can also be applied to conditional effects, for example, to test whether a parameter set differs across males and females. Moreover, LG-Syntax implements an option to obtain *user-defined Wald tests* for linear contrasts between parameters which should be defined in a separate text file.

LG-Syntax provides the option 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 also provides various special types of BVRs; that is, conditional, exact, longitudinal, and two-level BVRs. The latter two are also available in particular GUI models.

The Syntax allows specifying *user-defined expressions* of model parameters, either as part of the model equations or in a separate text file, for which one obtains point and standard error estimates. These functions can also be

used in user-defined Wald tests. Examples of applications in which this is useful include the computation of indirect effects, the computation of parameters with another coding, and the computation of predicted values for certain values of the latent and/or independent variables.

In models for categorical dependent variables with partially missing values, goodness-of-fit statistics will typically assume MCAR missing data (Vermunt, 1997). In LG-Syntax it is possible to request goodness-of-fit chi-squared statistics under the MAR assumption.

For the more general types of models available in Syntax, LG-Syntax provides output that is more general than the GUI modules output. 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. Also implemented is an alternative version of the ProbMeans output, which reports the marginal estimated class proportion conditional on a single independent variable computed using the model probabilities instead of the posterior probabilities. 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 7.2 for an example.

An alternative way of classifying new cases into latent classes that will work in most Cluster and DFactor-like models involves either requesting the scoring equations as part of the output or computing the scoring equations using the Step3 option (using either GUI or Syntax), and saving these in SPSS, R, or generic syntax format (Vermunt and Magidson, 2025). The scoring can then easily be performed outside LatentGOLD.

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, diagonal, offdiagonal coding, weight, prior, intercept, correlation and ar1, 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
- prior: defines the parameters of the Dirichlet priors for model probabilities
- diagonal and offdiagonal: define diagonal and off-diagonal parameters for associations between variables having the same number of categories
- intercept: indicates that the effect concerned is actually the intercept term
- correlation and ar1: defines a special structure for residual covariances between continuous dependent variables
- missingvaluedummy: generates a term consisting of a missing value dummy for the independent variable concerned
- categorical: treating a numeric independent variable as categorical/nominal in the term concerned
- numeric: treating a nominal independent variable as numeric in the term concerned
- ers: defines a parameter for the extreme responses of an ordinal dependent variable
- mrs: defines a parameter for the mid response(s) of an ordinal dependent variable
- pco: defines a partial correlation structure for the (residual) covariance matrix of continuous latent variables measured with continuous indicators (or factor scores)

- 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.
- The use of the classification EM algorithm in any type of latent class model.
- Performing asymptotic *power computations* for (user-defined) Wald tests and for likelihood-ratio tests between nested models.
- The use of the Lanza-Tran-Bray (LTB, 2013; see also Bakk, Oberski, and Vermunt, 2016) type of analysis for distal outcomes, possibly with bootstrap standard errors.
- A general method for standard error correction in stepwise estimation. Bakk, Oberski, and Vermunt (2014) used the approach to adjust of standard errors in the third-step model of a three-step latent class analysis by taking into account that this model uses the parameter estimates from the step-one model, which themselves have a particular variance-covariance matrix.
- In LC tree models, the Syntax contains various additional options for deciding whether a node should be split or not, in models with multiple latent variables it allows using the 'marginal' option for splitting, and it allows specifying the list nodes that should be split.

4 THE SYNTAX LANGUAGE

The following describes the overall organizational structure of a .lgs Syntax file; that is, the format that LatentGOLD uses when it saves possibly multiple model definitions of a Syntax session as a file with extension 'lgs'.

```
//LG 6.1//
version = 6.1
infile '<filename>' '<file format options>'
model
    <definition of 1st model>
end model
model
    <definition of 2nd model>
end model
model
    <definition of mth model>
end model
model
    <definition of last model>
end model
```

Users may construct their own file with such a structure or modify a saved Syntax file using a text editor, 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 1st model>, <definition of 2nd 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 LatentGOLD Syntax editor to reduce the possibility of errors introduced during the editing process. (See below for a description of the portion that *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 LatentGOLD 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.

This is the general structure of a LG Syntax model:

```
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, and user-defined expressions of 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,

title '<title>';

Example:

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 2.2) and may be used for different purposes including scoring additional files (see section 3.7). 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 2.2 for details).

4.1 OPTIONS SECTION

This section describes 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>
  <miterations=%d>
  <infmat=**hessian** | LM | BHHH | expected | numerical>
  <MstepNR>
  <NoMstepNR>
  <cem>
  <emfa>
  <expm=**eigen** | pade | taylor>;

- startvalues
  <sets=%d>
  <iterations=%d >
  <seed=%d >
  <tolerance=%d>
  <nriterations=%d >
  <pca>
  <anneal>
  <posterior>
  <partition>;
- bayes
  <latent=%f>
  <categorical=%f>
  <poisson=%f>
  <variances=%f>
  <ct=%f>
  <perclass>
  <percategory>
  <uniform>;
- 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>
  <norescale>
  <noignoreclassification>
  <aggregate>
  <bray>;
- tree

<**BIC** | AIC | AIC3>
<minentropyR2=%f>
<minN=%f>
<minBVR=%f>
<maxlevels=%d>
<maxnodes=%d>
<joint | marginal>;

- rotation <varimax | oblimin | geomin | target='filename'> | CFvarimax | CFquartimax | CFequamax | CFfacparsim | CFgeneral=<number between 0 and 1>
  <procrustes=%f | alignment=%f>
  <epsilon=%f >
  <delta=%f >
  <iterations=%d>
  <sets=%d>;
- output
  <parameters [= effect | last | first]>
  <profile [= longitudinal | posterior | LTB] >
  <probmeans [= **model** | posterior]>
  <marginaleffects>
  <bvr [= **marginal** | conditional | exact] >
  <bvrlongitudinal>
  <bvrtwolevel>
  <associationstatistics>
  <frequencies>
  <classification [= model | **posterior** | forward]>
  <estimatedvalues [= **model** | regression]>
  <iterationdetail(s)>
  <standarderrors [= **standard** | robust | fast | none | jackknife | npbootstrap [%d] | pbootstrap [%d] | expected | MCexpected | numerical>
  <SimulateParameters>
  <loadings>
  <reorderclasses>
  <ScoringEquations>
  <MARchi2>
  <identification [=%d]>
  <validationLL [= %d]>
  <predictionstatistics [= model | **posterior** | hblike | hblike2 | conditional]>
  <write | append ='filename'>
  <samplesizeBIC=%f>
  <ParameterCovariances ['filename']>
  <ParameterCorrelations ['filename']>
  <ignoreknownclass>
  <WaldTest='filename'>
  <WaldPower=%f>
  <LLdiff='H0model'>
  <LLdiffpower=%f>

<Vuong='H0model'>
<ScoreTest>[='filename']
<expressions='filename'>.
<writehessian='filename'>
<writegradients='filename'>
<writeparameters='filename'>
<writeinternalparameters='filename'>
<writeexemplarydata='filename'>
<writeBCH='filename'>
<writeclassificationerrors='filename'>
<writeclassificationerrorsproportional='filename'>
<writefactorerrors='filename'>
<writeSPSSsyntax='filename'>
<writeRsyntax='filename'>
<writeGensyntax='filename'>
<WriteRSGHB='filename'>;

- outfile '<filename>'
  <classification [= model | **posterior** | forward]>
  <logdensity>
  <factorscores=orthogonal | all>
  <bch>
  <prediction [=model | **posterior** | hblike | hlike2]>
  <individualcoefficients>
  <simulation [=%d] [seed=%d]>
  <imputation [=%d] [em | divisive | gibbs | dpp] [seed=%d]>
  <cooksD>
  <id = varname>
  <keep = varlist>;

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 setup during parameter estimation. The default is 'all' indicating that all available cores of the computer are used for the computations.

4.1.2 ALGORITHM

The algorithm subsection defines the technical settings referred to as 'Convergence Limits' and 'Iteration Limits' in GUI models. Additional options concern 1) the type of information matrix to be used in the Newton algorithm (infmat = hessian | BHHH | LM | expected | numerical), and 2) the number of M step iterations in the EM algorithm, 3) the use of Newton type of iterations within the M step of the EM algorithm (MstepNR), 4) the use of a classification EM approach, 5) the use of an EM algorithm within the M step for dealing with factor analytic models, and 6) the computation method for matrix exponentials.

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.

The default for the number of M step iterations in the EM algorithm equals 1, except for some cases where it is known that additional M step iterations may speed up convergence, such as in factor analytic models. In the latter cases this number is set to 5. The option 'miterations' allows modifying this setting. The option 'MstepNR' can be used to indicate LatentGOLD should maximize the expected complete data log-likelihood in the M step using a full Newton-Raphson method instead of the simpler unidimensional methods it uses by default. In certain types of models, LatentGOLD switches itself to multidimensional instead of unidimensional Newton updates within the M step of the EM algorithm. This can be suppressed using the new algorithm option 'NoMstepNR'.

The option 'cem' indicates that the specified model should be estimated using the classification EM algorithm (Celeux and Govaert, 1992). Similar to K-means clustering, this yields a hard-partitioning and it can in fact provide a K-means solution as a special case when all variables are continuous (Vermunt, 2011). The CEM-based hard-partitioning is applied to the latent variables at the highest level of the specified model, but it is never used for the dynamic states in latent Markov models.

De Roover, Vermunt, and Ceulemans (2022) described a rather general EM algorithm for mixture and/or multiple-group factor analysis, which can be further extended to include factor means (De Roover, 2021). This algorithm is activated in LG Syntax using the algorithm option 'emfa'.

In continuous-time (latent) Markov models, matrix exponentials are needed to obtain the transition probabilities and their derivatives from the transition intensities. The fastest method is based on an eigenvalue decomposition, but it fails when the transition intensity matrix concerned has equal eigenvalues (see, Vogelsmeier et al., 2019). Various alternative approaches have been proposed, and LG Syntax implements two of these; that is, the Padé approximation and the Taylor approximation, in both cases with scaling and squaring (Brančík, 2006, 2008; Moler and Van Loan, 2013). However, in practice, LatentGOLD users don't need to worry about this setting. By default, the faster 'eigen' method is used, but with an automatic switch to 'pade' for intensity matrices for which 'eigen' fails. This hybrid 'eigen-pade' approach turns out to work very well in practice.

4.1.3 STARTVALUES

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

The internal model structure of LG-Syntax models may be somewhat different from GUI models. Thus, a seed used in a GUI model will not necessarily obtain the same final estimates in the LG-Syntax.

There are three additional options compared to GUI models:

- The use of 'nriterations=%d' causes Newton iterations (in addition to EM) to be used in the starting values procedure.

- The option 'PCA' yields starting values for factor loadings based on a Principal Components Analysis (PCA) with the full sample and to which some random variation is added.
- The option 'anneal' invokes an annealing EM algorithm for the start sets in which the posterior class memberships are more fuzzy during the initial 100 EM iterations and where the fuzzyness is reduced every 5 iterations.
- The option 'posterior' is used to start the estimation with random values for the posterior class memberships (instead of random values for the parameters). The option is 'partition' is used to start with a random hard partitioning of the observation across the latent classes.

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 keywords (categorical, variances, latent, and poisson) are 0, that is, if one or more of the Bayes keywords are omitted from the options section of the Syntax, the Bayes constants are set to 0 for the omitted keywords.

Three additional Bayes options available compared to the GUI modules:

- 'perclass' indicates that the numbers specified with the other bayes options represent the amount of pseudo-data added to each class (instead of the total amount of pseudo-data)
- 'percategory' indicates that the number specified with the categorical option represents the average amount of pseudo-data added per category of the categorical variables, and
- 'uniform' indicates that the number specified with the categorical option is divided equally across categories, rather than based on the marginal distribution of the variable concerned.

Using all three options together yields a prior with a constant $\alpha = 1$ for each class-category combination.

LatentGOLD contains an experimental implementation of a gamma prior for the transition intensities in continuous-time (latent) Markov models, which is aimed at preventing that these become exactly 0 (their boundary value). This option is activated using the ct=%f statement as part of the bayes options.

4.1.5 MISSING

This subsection defines how the program should 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, where
    - L2 yields a bootstrap p-value for the L-squared goodness-of-fit statistic, while allchi2 yields bootstrap p-values for all goodness-of-fit statistics including the BVRs. The latter option is comparable with the “Bootstrap Chi^2” option in GUI models (either via the Model menu or the Output tab).
    - LLdiff yields a bootstrap p-value for the likelihood-ratio test comparing two models, an approach which is often referred to as a bootstrap likelihood-ratio test (BLRT). Using LLdiff='model name' (or %d) yields a likelihood-ratio test comparing the current (H1) model with a more restricted (H0) model called 'model name' (or with a number %d). This is comparable with the “Bootstrap -2LL Diff” option in GUI models obtained via the Model menu. When running models for a range of latent classes, one can also use LLdiff without specification on the H0 model. In that case, a BLRT will be performed for each pair of consecutive models with one class difference. This is equivalent to the Output option “Bootstrap -2LL Diff” in GUI models.
    - 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 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 then 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 parameters 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.
- By default, the posteriors specified when defining a step3 model are rescaled to sum to 1. This rescaling can be suppressed by using the keyword 'norescale' in the step3 option, which may be useful if one specifies a step3 model using only a subset of the original latent classes.
- By default, the assigned class membership is not used to obtain the classifications in a step3 model, which in most situations are not very relevant anyhow. That is, it is ignored when computing posterior classification probabilities and classification statistics. This setting can be modified using the step3 option noignoreclassification.

- The step3 option ‘aggregate’ can be used in step-three models for continuous latent variables. With this option, one indicates that the factor variances (needed to obtain the fixed loadings and residual variances for the factor scores) should be based on all observations when using the ‘dif’ option.
- With the step3 option ‘bray’ one requests the method proposed by Bray et al. (2019) which uses a single IPW for all classes, whereas in the default setting each class has its own IPW, which is basically the inverse of the propensity score for the class concerned (Lê, Clouth, and Vermunt, 2025). See also the ‘propensity’ option for latent variables.

In more complex step3 models, only the modal-ml and modal-none combinations can be used. This applies to latent Markov models and multilevel LC models.

4.1.9 LC TREE OPTIONS

LG Syntax implements extended options for LC tree modeling. Whether or not to split a node (yes or no) will typically be decided based on the increase in LL between the 1- and 2-class model. The minimally required LL difference cutoff can be defined by the user (option minLLdiff=%f) or be derived from the BIC, AIC, or AIC3 (options BIC, AIC and AIC3). Another criterion that one can select as a cutoff is the minimum entropy R-squared value threshold for the 2-class model to achieve in order to split into these 2 classes (minentropyR2=%f). Alternative criteria for not splitting can also be specified. These include when the sample at a node is too small (minN=%f), when the largest BVR in the 1-class model is too small (minBVR=%f), when a certain maximum number of levels is reached (maxlevels=%d), or when a certain maximum number of nodes is reached (maxnodes=%d).

A final tree option concerns the way the splitting and modeling at the next levels is performed when a model contains two or more discrete latent variables. When using the default option ‘joint’, at the root node the child nodes are formed by the joint classes obtained by crossing the multiple latent variables. Following the construction of the root level, for the next levels, 2-class models are estimated and binary splits are performed (in fact, the number of categories is set to 2 for the tree latent variable and to 1 for the other discrete latent variables). When using the option ‘marginal’, the splitting is based on the marginal posteriors of the tree latent variable. The model which is estimated at each of the nodes is the original model with multiple latent variables, but with the number of categories of the tree latent variable set to 2.

The following is an example of the specification of the tree options, with the default settings selected:

```
tree BIC minentropyR2=0 minN=0 minBVR=0 maxlevels=5 maxnodes=500 joint;
```

4.1.10 FACTOR ROTATION OPTIONS

The LG Syntax module implements factor rotation options which can be used to obtain interpretable and identified exploratory factor analysis (EFA) solutions. The rotation criteria implemented to achieve a simple structure are varimax, oblimin, geomin, and target rotation, as well as various rotation criteria from the Crawford-Ferguson (CF) family. Recent work by De Roover and Vermunt (2019) extended the approach of Jennrich (1973, 2002) to multiple- group and mixture EFA. In such applications, the rotation criterion consists of two terms, one rotating to simple structure within groups/classes and one rotating to agreement between groups/classes. For the latter, one can use either the procrustes or the alignment criterion. Users need only specify the weight (w) to be assigned to the agreement criterion, and the weight of the simple structure criterion is then set to the remainder (1-w). When using one of the agreement options in combination with free factor variances in all groups/classes, LatentGOLD will restrict the factor variances to have a mean of 1 across groups/classes.

With target rotation, the target structure should be specified in a text file. Typically, only the 0 loadings are relevant for the target, but also other values of target loadings may be used. Use of either the missing value code '.' or the number -99 indicates that the loading concerned does not contribute to the target criterion. A separate target can be specified for each group/class (they appear stacked in the file), but when the file contains the target only for one group/class, it is assumed that the target is the same for all groups/classes.

The option 'epsilon' refers to the small number appearing in the formula of the alignment criterion, which by default equals 1.0e-12. The option 'delta' allows changing the small number appearing in the geomin criterion, which by default equals .0001 with less than 3 factors, .001 with 3 factors, and .01 otherwise. The option 'iterations' can be used to increase the maximum number of Fisher-scoring iterations used for the rotation (the default is 25). The option 'sets' can be used to indicate the rotation should be performed using multiple starting values for the rotation matrix (to prevent local minima of the rotation criterion). The default value is 1.

4.1.11 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.
- Paired Comparisons, which are Wald tests comparing pairs of classes.
- Loadings for Cluster- and DFactor-like models, which are obtained using a linear approximation of the class-specific response probabilities (Vermunt and Magidson, 2005).
- 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. As in GUI models, Profile also contains an interactive plot.
- Profile-Posterior gives the part of the GUI Profile which is obtained using the posterior class-membership membership probabilities. This output section contains information on (inactive) covariates (called independent variables Syntax models).
- 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-posterior (called Probmeans in GUI models), which also yields average posterior probabilities/means also for group-level and dynamic (Markov-type) latent variables.
- Frequencies.
- Bivariate Residuals, including the 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 (if present).
- IterationDetails.
- Prediction statistics – which are basically (pseudo) R-squared values - are provided for the dependent variable(s) in Regression and Choice (posterior, model, and hblike), and in Step3-Dependent (conditional). In Syntax, one can obtain prediction statistics is for any type of model. Note that the posterior and conditional options differ in the way the posterior probabilities are used in the computation of the loss function (e.g. the squared error). That is, in posterior we average the class-specific predictions (using the posteriors as weights) and compare these with the observed value, whereas in conditional we compare the class-specific predictions with the observed value and average the resulting values of the loss function (using the posteriors as weights).
- ScoringEquations for Cluster- and DFactor-like Syntax models (models without id variables), which are the logistic equations which can be used to compute the posterior classification probabilities.
- WriteSPSSSyntax, WriteRSyntax, and WriteGenSyntax can be used to write the scoring equations to file, that is, as SPSS syntax, as R code, or in a more generic (C-code like) form.
- For Choice models, there is an output option WriteRSGHB='filename'. This yields an R script and a data file that can be analyzed with the RSGHB package in R.

Output sections/options specific for Syntax are:

- Marginal effects, which are partial effects on probabilities or expected values (Long, 1997; Long and Freese, 2014). LG Syntax implements *average marginal effects* (as well as their standard errors, z values, and p values) for all types of regression parameters, including the logit coefficients for covariate effects on latent classes or transition probabilities.
- Profile-LTB, which also yields class-specific means and variances for continuous distal outcomes used as covariates (Lanza, Tran, and Bray, 2013), as well as a Wald test for the equality of the means across classes. As shown by Bakk, Oberski, and Vermunt (2016), correct standard errors and tests can be obtained by combining the LTB approach with bootstrap standard errors, which is achieved with the output option standarderrors=npbootstrap.
- 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.

Additional output options include the possibility

- to request that the latent classes are reordered from large to small as is done in GUI models (reorderclasses option),

- to request associationstatistics, which are overall and bivariate chi-squared statistics under independence, which are, in fact, measures of the strength of the overall and bivariate associations between the observed variables. These observed associations in the data are compared with their simulated values under the model to be tested (Nagelkerke, 2018; Kollenburg, Mulder, and Vermunt, 2018),
- to use the option SimulateParameters invoking a resampling method in which parameter sets are sampled directly from the MVN distribution defined by their point estimates and their estimated asymptotic covariance matrix. The relevant functions of parameters are computed with each sample and their SDs across samples serve as SE estimates. Those functions can be the standard LatentGOLD output sections Profile and EstimatedValues, but can also be user-defined expressions (see below),
- to obtain adjusted chi-squared statistics under MAR missing data (Vermunt, 1997). This is achieved with the MARchi2 option. It should be noted that this option should not be used in tables with more than 100,000 cells, because otherwise the necessary estimation of the saturated model may take very long.
- 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 (nested) within Parameters, where EPC(other) refers to the expected parameter changes for the (other) free parameters. The specification of the filename is only needed when one wishes to define one or more sets of parameters for which a simultaneous test should be performed,
- to specify user-defined Wald tests for linear restriction on the model parameters,
- to specify user-defined expressions of parameters in a separate file (rather than as part of the equations section),
- to compute the power or the required sample size for the Wald tests appearing in Parameters (Gudicha, Tekle, and Vermunt, 2016). WaldPower=n indicates one the sample size to achieve a specified power (if n<1) or the power achieved by a specified sample size (if n>1),
- to compute the asymptotic power of the likelihood-ratio test using the procedure described in Gudicha, Schmittmann, and Vermunt (2016), which involves defining and running H1 and H0 models using either an exemplary data set or a large simulated data set under the H1 model (in which one defines LLdiff='H0model'). LLdiffPower=n indicates the sample size to achieve a specified power (if n<1) or the power achieved by a specified sample size (if n>1),
- to obtain Vuong likelihood-ratio tests comparing the current (H1) model with the specified H0 model (Vuong, 1989; Merkle, You, and Preacher, 2016, Schneider, Chalmers, Debelak, and Merkle, 2020; Vermunt, 2024). Note that as in GUI models Vuong likelihood-ratio tests are obtained automatically when running models for a range of classes, in which case consecutive models with one class difference are compared. The latter yields the test proposed by Lo, Mendell, and Rubin (2001), which is often referred to as the VLMR or LMR test.
- 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, or 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 (minus inverse) information matrix (writehessian), the gradient contributions (writegradient), all data patterns with their expected frequencies (writeexemplarydata), the parameters (writeparameters), the internal parameters (writeinternalparameters), the weights used in the 3-step bch adjustment (writebch), the classification error logits needed for the 3-step with ml adjustment (writeclassificationerrors and writeclassificationerrorsproportional), and factor loadings and residual variances for 3-step SEM (writefactorerrors).

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 7.11.

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 BVRs for two-level and longitudinal data sets, which are also available in LG GUI Markov and Multilevel models.

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 option hblike2 differs from hblike only in models with a scale factor; that is, in hlike2 the scale model is applied before averaging the linear term over classes while in hblike it is applied after averaging.

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 7.12.

When using the option 'ignoreknownclass', classification statistics and classification-posterior output will be computed without using known-class information.

A GUI output section that is not available in Syntax is the Random-Effects output section from the Regression module.

4.1.12 OUTFILE OPTIONS

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 |hblike2]' are the same as discussed above for classification output and prediction statistics. The option classification = forward is an option for Markov models. Note that classification = model and classification = posterior are also referred to as *covariate* and *standard classification*, respectively.

The outfile option 'logdensity' allows saving the class-specific log $P(\mathbf{y}|x)$ values to an output data file. The saved log $P(\mathbf{y}|x)$ values can be used for a stepwise LC analysis with the Bakk and Kuha (2018) approach; that is, by using logdensity=(<list of logdensities>) in the definition of the latent variable(s) concerned.

The outfile option 'bch' can be used in conjunction with classification=posterior. This yields BCH weights for modal and proportional assignment in the output data file in addition to posterior class membership probabilities. The BCH weights can be used to perform a BCH-type step-three analysis without the use of the LatentGOLD step3 procedure.

Two other new options which can be used in conjunction with classification=posterior are 'factorscores=orthogonal' and 'factorscores=all', yielding factor scores ignoring factor covariances and class-specific factor scores, respectively. The 'orthogonal' option is relevant for three-step models with continuous latent variables, which require that the factor scores are based on uncorrelated factors. The 'all' option may be relevant in mixture factor analysis models, for example, if one wishes to use the factor scores in a separate three-step model for each mixture component.

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' 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. When using simple latent class models for categorical dependent variables, one can also obtain multiple imputations using a divisive Latent Class procedure, Bayesian estimation with Gibbs sampling, and Bayesian estimation using a Dirichlet process prior (Van der Palm, Vermunt and Van der Ark, 2016b; Vidotto, Kaptein, and Vermunt, 2015, Vidotto, Vermunt, and Van Deun, 2018, 2020). These experimental options are invoked with the keywords 'divisive', 'gibbs', and 'dpp' respectively in the outfile option 'imputation'. For the divisive procedure one should use a 2-class model, for the gibbs procedure a LC model with the number of classes one wishes, and for the dpp procedure a model with a large number of classes.

The 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 file (the inpfile). This prevents the need to merge the LatentGOLD classification/prediction output file with the original file prior to performing subsequent analyses, for example, a step-three analysis.

4.1.13 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 reorderclasses;
   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

These are the keywords for specifying the id variables:

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

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

In latent Markov models, multiple records 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 FREQUENCY WEIGHTS

These are the keywords for specifying frequency 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

These are the keywords for specifying the dependent variables:

- dependent <varname | (varlist)>
  <nominal | **ordinal |** continuous | poisson | binomial | cumlogit | probit | loglog1 | loglog2 | seqlogit1 | seqlogit2 | log | lin | gamma | beta | vonmises | mpoisson | multinomial | dirichlet | exponential | weibull| gompertz | loglogistic | lognormal | cox >
  <%d>
  <overdispersed>
  <censored | truncated>
  <upper= varname | (limits of time intervals)>
  <zeroinflated>
  <event=varname>
  <t0=varname>
  <aft>
  <exposure= varname | %f>
  <coding= effects | first | last | %d>
  <rank=%d>
  <score=(%f, %f,…, %f)>
  <varweight=%f>
  <ignoreclassification>,
  next dependent variable, …;

This option is used to define the dependent variable(s) in the model. Note that *dependent* is used as a generic term; that is, it includes what is referred to as the dependent variable in the Regression module and indicators in

the Cluster and DFactor modules, but also what is sometimes referred to as response variables, outcome variables, etc.

Note that it is possible to specify the features for a list of dependent variables defined within '(…)' rather than for a single dependent variable.

The keyword <nominal | **ordinal** | continuous | poisson | binomial | cumlogit | probit | loglog1 | loglog2 | seqlogit1 | seqlogit2 | log | lin | gamma | beta | vonmises | mpoisson | multinomial | dirichlet | exponential | weibull | gompertz | loglogistic | lognormal | cox > 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, mpoisson), logistic for counts (binomial, multinomial), cumulative ordinal logistic (cumlogit), ordinal probit (probit), ordinal log-log (loglog1 or loglog2), sequential logit (seqlogit1 or seqlogit2), loglinear and linear for cumulative probabilities (log and lin), log-linear gamma (gamma), logit (beta, Dirichlet), linear von Mises (vonmises), or survival models (exponential, weibull, gompertz, gamma, loglogistic, lognormal, cox). 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, binomial and multinomial counts, yielding negative-binomial (or negative-multinomial), beta-binomial, and Dirichlet-multinomial models, respectively. The 'censored' keyword can be used with continuous dependent variables, yielding tobit regression models, and can be combined with the 'upper' option yielding interval censored models. The 'truncated' option can be used with continuous, Poisson, mpoisson, binomial, and multinomial, 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 options 'upper= (limits of time intervals)', 'event=varname', 't0=varname', and 'aft' are specific for survival models. That is:

- 'upper= (limits of time intervals)' can be used in conjunction with 'exponential' to define a piecewise exponential survival model, where the list defines the upper limits of the time intervals
- 'event=varname' is used to define the 0-1 variable indicating whether an event occurred (=1) or whether the observation was right censored (=0). When omitted, LG assumes everyone experienced the event at the specified time.
- 't0=varname' is used to define the time of entry into the risk set (or the left censoring time)
- 'aft' is used to switch from the default log-linear hazard parametrization for exponential, Weibull, and Gompertz, piecewise exponential, and Cox models. Gamma, lognormal, and loglogistic models are always in accelerated failure time form.

The available regression models are:

- nominal: binary or multinomial logistic
- ordinal: binary or adjacent-category ordinal logistic
- continuous: normal linear
- continuous censored: tobit (or censored normal) linear
- continuous interval censored: interval-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
- log: binary or cumulative ordinal log-linear model
- lin: binary or cumulative ordinal linear model
- 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
- exponential, weibull, gompertz,gamma, loglogistic, lognormal, and cox: log-linear hazard rate and/or accelerated failure time models

In addition, the available multivariate response regression models are:

- mpoisson: log-linear multivariate Poisson
- mpoisson truncated: log-linear zero-truncated multivariate Poisson
- mpoisson overdispersed: log-linear negative-multinomial
- mpoisson overdispersed truncated: log-linear zero-truncated negative-multinomial
- multinomial: multinomial logistic
- multinomial truncated: zero-truncated multinomial logistic
- multinomial overdispersed: logistic dirichlet-multinomial
- multinomial overdispersed truncated: logistic zero-truncated dirichlet-multinomial
- dirichlet: logit Dirichlet model for multivariate continuous responses summing to 1

In multivariate response models, the multiple variables from the data file should be specified within '(…)', that is, using the '(varlist)' option.

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 and multinomial 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 LatentGOLD '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 varweight is useful if I one wishes to downweigh the influence of a particular (set of) variable(s) relative to the other variables. For example, in a LC model with both continuous and categorical indicators, one may wish to reduce the influence of the continuous indicators on the clustering, which may be achieved by giving them a lower weight, say of .5 instead of 1.

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,
          (y51 y52 y53) multinomial exposure=tot5;
```

4.2.4 INDEPENDENT VARIABLES

These are the keywords for specifying the independent variables to be used in a model:

- independent <varname | (varlist)>
  <nominal | **numeric** | bspline [degree=%d] [knots=%d] [knots=(knot locations)]>
  <inactive>
  <coding= effects | first | last | %d>
  <rank=%d>
  <score=(%f, %f,…, %f)>
  <propensity=(varlist)>,
  next independent variable, … ;

Note that *independent* is a generic terms; 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*.

Note that it is possible to specify the features for a list of independent variables defined within '(…)' rather than for a single independent variable.

For independent variables, the scale type can be defined to numeric, nominal, or bspline. 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.

The 'scale type' bspline indicates the effects of the independent variable concerned should be modeled using B-spline functions (Boor, 1978), for which one may wish to specify the degree (default is 3) and the (number of) knots (default is 0 inner knots). When specifying the number of inner knots, these will be equally spaced between the lowest and the highest observed value of the independent variable concerned, which serve as the outer knots. When specifying the knots' locations, one should not only provide the inner knots but also the two outer knots.

An example of the specification of the independent variables is:

independent gender nominal, education, age bspline degree=3 knots=2, region nominal inactive;

LatentGOLD allows specifying propensity scores corresponding to an independent variable that serves as treatment or exposure variable. This is an alternative to the ipw option described below. The specified propensity scores are used to compute the IPWs. Sample Syntax is:

independent treatment propensity=(prop1 prop2 prop3);

4.2.5 LATENT VARIABLE INFORMATION

These are the keywords to define the latent variables which are part of the model:

- latent <latvarname>
  <**nominal** | ordinal | continuous>
  <%d> or <%d>:<%d>
  <group | **case** | dynamic>
  <coding= effects | first | last | %d>
  for continuous: <nodes =(%f, %f,…, %f)> <distribution=(%f , %f, …,%f)> <halfnormal>
  for ordinal: <score =(%f, %f ,…, %f)>
  for nominal / ordinal: <coding= [effects | first | last | %d]>
  for dynamic: <dt>
  <tree [split=(list of nodes)]>
  <knownclass varname[=(first value:0|1 0|1…0|1,… , last value:0|1 0|1…0|1)]>
  <posterior = (varlist) [dif=<independent variable> | (list of independent variables)]>
  <logdensity = (varlist)>
  <propensity=(varlist)>,
  next latent variable, …;

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

<**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 (<%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, <%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

<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

<nodes =(%f, %f ,…, %f)>

and

<distribution=(%f , %f, …, %f)>

When using the option “halfnormal” for continuous latent variables, the negative Gauss-Hermite quadrature nodes are flipped, implying this option will yield a half normal distribution with non-negative values only.

The options <score=(%f, %f ,…, %f)> for ordinal latent variables and <coding=<effects | first | last | %d> for ordinal and nominal latent variables have the same functionality as the same option in dependent and independent.

The ‘dt’ option is relevant for dynamic latent variables in continuous-time latent Markov models. It allows indicating that one or more of the nominal or ordinal dynamic latent variables should be modeled as in a discrete-time (dt) model; that is, with transition probabilities modeled using standard logit equations.

An example of the use of the latent variable specification:

latent gclass group 3 coding=last, cfactor continuous, dfactor ordinal 3 score=(0, 1, 3);

The keyword ‘tree’ is use to indicate which latent variable is the one that should be used for splitting in LC tree models (van den Bergh, Schmittmann, and Vermunt, 2017; van den Bergh, Kollenburg, and Vermunt, 2018) . It is also possible to construct the tree exactly as one wishes, that is, by specifying which nodes should be split. This is achieved with the option split=(list of nodes), for example,

latent Cluster nominal 3 tree split=(1 3 31);

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 latent 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-1 analysis to an output data file. This option can be used for any of the ordinal/nominal latent variables in the model. The same option can be used to specify step-three models with continuous latent variables. In that case, one should provide the names of the variables in the data file containing the factor scores and the standard errors of the factor scores. The keyword dif=<independent variable> with a single DIF variable or dif=(<list of independent variables>) is used when performing a step-three analysis taking into account differential item functioning (see Vermunt and Magidson, 2021).

The option ‘logdensity=(varlist)’ is needed for stepwise LC modeling using the Bakk-Kuha approach (Bakk and Kuha, 2018). Rather than defining the number of categories of the ordinal/nominal variable concerned, one specifies the variables in the input data file containing the log-densities for the K classes; that is, the variables obtained by writing the log-density information from the step-1 analysis to an output data file. This option can be used for any of the ordinal/nominal latent variables in the model.

The option ‘propensity=(varlist)’ can be used in a distal outcome step3 analysis in which class membership serves as exposure or treatment (Bray et al., 2019; Lê, Clouth, and Vermunt, 2025). The specified propensity scores are used to compute the class-specific IPWs. Note that this option should always be combined with the BCH bias-adjustment method.

4.2.6 SURVEY OPTIONS

The five keywords of interest are:

- stratumID <varname> <certain | scaled | centered>;
- psuid <varname>;
- samplingweight <varname> <rescale> <ipw>;
- populationsize <varname>;
- rsweights <varlist>;

These allow specifying 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 to sum to the sample size. The option ‘ipw’ can used to indicate that the specified sampling weights are, in fact, inverse propensity weights, which is mainly useful in step3 models (see Clouth, Pauws, Mols, and Vermunt, 2022).

Replicate sampling weights available for variance estimation can be defined with the rsweights keyword.

The stratumid options certain (default), scaled, and centered refer to three alternative ways to deal with strata consisting of a single PSU. Details on these options are provided in the LatentGOLD 5.1 upgrade manual and in the technical guide.

4.2.7 OPTIONS FOR MARKOV MODELS

Two options which are specific for Markov models are:

- timeinterval <varname > [obstype=<varname>]
- 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, this option is also used to indicate that the specified model is a continuous-time Markov model. While by default the data are assumed to represent snapshots of the process at arbitrary times, observations can also represent exact times of transition (as in survival analysis), exact death times (the time entering in the absorbing state concerned, but it is unknown from which origin state), or a mixture of these. With the observation-type option, one can indicate what kind of observation it concerns, where a 1 indicates a time interval of the snapshot, 2 an exact transition time, and 3 an exact death time.

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. When using this option one obtains log-likelihood, classification, and prediction statistics based on the holdout cases in the n 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 an 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:

<varlist> <- <(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 (note that it can also be a list of variables). 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)> <varlist> < | 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 variable ‘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)> <varlist> <-> <varlist> <| varlist>;

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

Example:

response1 <-> response2 | 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

catresp1 <-> catresp2 <-> catresp3;

4.3.4 LOG-LINEAR SCALE FACTOR EQUATIONS

For nominal, ordinal, choice-type (choice, ranking, best-worst, and rating), continuous, and Poisson 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,

<varlist> <<- <(label)> <varlist> < | varlist> + … +… ;

An example of a scale factor equation is:

response <<- Class + 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:

response1 <- 1 + class;
response2 <- 1 + class;
response3 <- 1 + class;
response4 <- 1 + class;

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

response1 response2 response3 response4 <- 1 + class;

or reduced even further using the '-' option:

response1 - response4 <- 1 + 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.

LG-Syntax allows for a special use of a list of variables at the LHS and RHS simultaneously. Suppose we have 4 latent variables called class1, class2, class3 and class4, and each of which affects one of the 4 dependent variables. This can be specified in a compact manner as follows (with the list at the RHS between "{…}"):

response1 - response4 <- 1 + {class1 – class4};

which is equivalent to saying:

response1 <- 1 + class1;
response2 <- 1 + class2;
response3 <- 1 + class3;
response4 <- 1 + class4;

4.3.6 PARAMETER RESTRICTIONS

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

- Equality: <label><[%d,%d]> = <label><[%d,%d];
- Fixed value: <label><[%d,%d]> = %f | {…} | <filename>;
- Ratio: <label><[%d,%d]> = %f <label><[%d,%d]> ;
- Monotonicity: <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

```
class <- 1;
y1 <- (a) 1 | class;
y2 <- (b) 1 | class;
y3 <- (c) 1 | 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; 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; 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; 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:

response <- 1 | gender + (b) time | 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 '=':

```
<label><[%d,%d]> ~= <%f> | {…} | 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 RESTRICTIONS OR STARTING VALUES FOR A LIST OF PARAMETERS

Starting values and constraints can also be specified for multiple sets of parameters simultaneously. A few examples are:

```
a[1,1] b[1,1] c[1,1] ~= {<startvalues>};
```

which reads starting values for these parameters,

```
a - c d ~= 'name of file with starting values';
```

which reads starting values for parameters a to c and d,

```
a b c = {< fixed values >};
```

which provides fixed values for parameters a, b, and c,

```
a b c = 2;
```

which is a shortcut for a = 2; b = 2; c = 2;

```
a b c = d;
```

which is a shortcut for a = d; b = d; c = d;

```
a b c = +;
```

which indicates a, b, and c should be non-negative.

Thus, at the left-hand side one can specify a *list* of parameters. The right-hand side remains as usual. If the number of specified fixed or starting values is fewer than the total number of parameters implied by the list, LatentGOLD will return to the first value after reading the last value and continue doing this until the complete sequence of parameters is processed.

4.3.9 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.

4.3.10 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 ~dif
- ~nominal or ~nom
- ~ordinal or ~ord
- ~squared or ~sqr

- ~categorical or ~cat
- ~numeric or ~num
- ~full or ~ful
- ~diagonal or ~dia
- ~offdiagonal or ~off
- ~ers
- ~mrs
- ~ar1
- ~correlation or ~cor
- ~pco
- ~wei or ~weight
- ~int or ~intercept
- ~mis or ~missingvaluedummy
- ~prior or ~pri

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 options ~categorical and ~numeric can be used to change the scale type of and independent variable in the term concerned. That is treat a numeric independent variable as categorical/nominal or a nominal independent variable as numeric.

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 options ~dia and ~off are used to define diagonal and off-diagonal parameters in associations consisting of variables with equal numbers of categories. The off-diagonal option is similar to the ~tra and ~err specifications, but these work only with latent-latent and latent-dependent associations, respectively, while ~off option also works with independent-dependent and dependent-dependent associations.

The options ~ers and ~mrs can be used in response style models for ordinal indicators, where ~ers defines a parameter for the extreme responses and ~mrs a parameter for the mid response(s).

The options ~ar1 and ~cor are used to model the residual covariance structure for continuous dependent variables. The ~ar1 option yields an autoregressive covariance structure, which may be useful in longitudinal data applications. The ~cor option gives covariances in terms of correlations times the standard deviations of the two variables involved, which is useful if one wishes to impose constraints on correlations rather than on covariances.

The option ~pco modifies the parametrization of the (residual) covariance matrix of continuous latent variables with continuous indicators. More specifically, it indicates this matrix should be modeled using partial correlations, which makes it possible to perform tests and impose restrictions on partial correlations instead of covariances.

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, and in models for counts. 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 ~wei option can also be used in linear models for continuous dependent variables, were it is used as an offset, thus as an additive instead of multiplicative term.

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).

The ‘parameter’ option ~pri is used to indicate one wishes to use specific values for the Dirichlet parameters of the prior distributions for the model probabilities for nominal and ordinal latent and dependent variables. These are specified as ‘fixed values’. The priors specified in this way override the settings used for the bayes options.

4.3.11 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),
- scoring of 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 continuous 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.

4.3.12 READING FIXED VALUES AS WELL AS THEIR VARIANCE-COVARIANCE MATRIX FROM A TEXT FILE

Bakk, Oberski, and Vermunt (2014) discussed the adjustment of standard errors in the third step model of a three-step latent class analysis by taking into account that this model uses the parameter estimates from the step-one model, which themselves have a particular variance-covariance matrix. In fact, they use a rather general method for standard error correction in stepwise estimation. Although not straightforward to use, this general procedure is also available in LG-Syntax.

The procedure is invoked by defining fixed values for either a single set or a list of (sets of) parameters, and specifying the name of the file from which the fixed values can be read. However, the specified file may contain more information than just the fixed values; that is, it may contain the following five items:

- Values of the fixed parameters. This is required.
- Number of original parameters in the step-1 model and the Jacobian matrix, which is the matrix of first derivatives of fixed parameters towards the step-1 parameters. This is optional. It can be used when the fixed parameters are functions of the step-1 parameters.
- Minus inverse Hessian matrix of the step-1 parameters. This is required.
- Covariance matrix of the step-1 parameters. This is optional, It is used when another kind of SE estimator is used in step-1 than the one based on minus inverse Hessian.
- Gradient contributions of all observations for the step-1 parameters. This is optional. It is used for the second order correction described by Bakk, Oberski, and Vermunt (2014). When using this option one should make use of a caseid in both runs to make sure that the sorting of the records coincides between the two estimation steps.

Each of these five items is preceded by a keyword as follows – Step1Parameters, Step1Jacobian, Step1Hessian, Step1Variances, and Step1Gradients, respectively – and followed by the numbers. To see how the file is structured one can use the output option writeclassificationerrors= ‘filename’. This yields a file with classification error logits as Step1Parameters and each of the other 4 items discussed above.

4.3.13 UNALLOWED MODEL SPECIFICATIONS

Certain model specifications are not allowed because they yield meaningless models, or models that are not yet implemented in LatentGOLD. When such model specification statements are encountered, the program will report an error message indicate 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 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 LatentGOLD 6.1. 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, except for continuous latent variables with continuous indicators (or factor scores as single indicators) for which LG-Syntax allows specifying a true SEM model

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 this 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.

4.3.14 USER-DEFINED EXPRESSIONS

Often, we are interest in functions of model parameters. Sometimes, getting their point estimates is enough, but in other situations we may also wish to quantify their uncertainty (their SEs) or use them in univariate z or multivariate Wald tests. Examples of applications in which this is useful include the computation of indirect effects, the computation of parameters with another coding, and the computation of predicted values for certain values of the latent and/or independent variables.

LatentGOLD Syntax implements an option for specifying user-defined expressions or functions. These expressions may contain parameter labels, names of other expressions, numbers, operators (+, -, /, *, and ^), and mathematical functions like sqrt(), log(), and exp(). These expressions can either be defined at the end of the equations section, together with the parameter constraints, or in a separate file defined in the output section using expressions=<filename>. The general form of an expression appearing in the equations section is:

%<name> = <expression>;

The name of an expression should be unique in the sense that it was not used as variable name or parameter label in the model specification, and when used in the equations section it should start with the symbol '%'. More specifically, if the program encounters a line starting with a % in the equations section, it recognizes it is an expression. Within an expression, it is possible to use the name of other expressions (without the symbol '%').

This is sample Syntax of a model with an indirect effect of a 4-class latent variable Cluster on a continuous outcome Y via a continuous mediating variable X (for which we have two copies in the data file, namely Xdep and Xindep):

```
equations
  Cluster <- 1;
  Xdep <- 1 + (a) Cluster;
  Y <- 1 + (b) Xindep;
  Xdep;
  Y;
  %c1 = a[1,1]*b[1,1];          // indirect effect for Cluster 1
  %c2 = a[1,2]* b[1,1];         // indirect effect for Cluster 2
  %c3 = a[1,3]* b[1,1];         // indirect effect for Cluster 3
  %c4 = - c1 - c2 - c3;         // indirect effect for Cluster 4
```

As can be seen, we compute the products of the 3 parameters for the effect of Cluster on X and the effect of X on Y. Because of the effect coding scheme, the indirect effect for Cluster 4 equals the negative of the sum of the indirect effects for Cluster 1, 2, and 3. Note that the above model could be a step-3 model using latent classes obtained from a step-1 analysis.

Version 6.1 implements extended Syntax which allows defining a vector of expressions and summing the elements of a vector. This can best be illustrated using the indirect effects example from above, which can also be defined as follows:

```
%c[1:3] = a[1,1:3]*b[1,1];    // indirect effects for Clusters 1-3
%c[4] = - c[+];               // indirect effect for Cluster 4
```

Thus we define c[1], c[2], and c[3] using a single equation, and sum these using the a "+" as index to compute c[4]. After applying the second expression, the vector c will have 4 entries.

Nested within the Parameters output, the program will report the values of the user expressions, as well as their SEs and corresponding z tests. Note that by default the tests are based on asymptotic standard errors, but it is possible to use any of the other standard error estimators implemented in LatentGOLD.

4.4 SYNTAX SPECIFIC FOR CHOICE MODELS

LatentGOLD Syntax contains various keywords which are specific for choice models. Below, we present these Syntax commands which are used to define the file format, and which are part of the options, variables, and equations sections of the Syntax. The upgrade manuals for LatentGOLD Choice 5.0 and LatentGOLD 6.0/6.1 describe these commands in more detail, as well as provide technical information on the underlying mathematical equations.

4.4.1 CHOICE FILE FORMATS

An important part of the syntax in choice models concerns the section specifying the data file format. This section appear between the `options' and `variables' sections. Choice data can be in the form of either a single file or three files. That is, one will use the setting

- choice=1;

or

- choice=3
  alternatives=<filename> id=<varname>
  choicesets=<filename> id=<varname>;

In the 1-file format, alternatives and their attributes are in the response file (the infile), that is, each record is a choice alternative. The choice-set id variable connects the alternatives belonging to the same choice set. The dependent variable takes on the value 1 for the first choice, 2 for the second choice, etc., and equals 0 for the non-selected alternatives.

As can be seen, for the 3-file format, besides the response file (the 'infile') containing information on the selected alternatives, a file containing the alternatives and their attribute values and a file defining the choice sets and their alternatives should be specified. The three files are linked by using the alternatives' ids in the choice set file, and the choice sets' ids in the response file.

4.4.2 CHOICE OPTIONS

Choice-specific keywords which are part of options section of the Syntax are the output options setsprofile, setsprobmeans, and importance, and the outfile option individualcoefficients, which in best-worst models also yields the individual best-worst scores.

4.4.3 CHOICE VARIABLES

Variables which are specific for choice models are:

- the choice-set id variable
- the replication scale variable
- the specific 'scale types' for the dependent variable concerned, and
- the attributes, which are alternative-specific independent variables.

The choice set number of a choice is specified as follows:

- choicesetid <varname>

In the 1-file format, the choicesetid connects the records in the response data file which belong to the same choice set. In the 3-file format, it links the choices in response data file to the choice sets defined in the choice-sets file.

The replication scale is a constant by which the linear term in the model for the choice concerned is multiplied. For example, a value of -1 implies that choice concerned is a worst choice instead of a best choice. However, because best-worst models are part of LatentGOLD 6.1, this option has become less relevant. The keyword is used as follows:

- repscale <varname>

These are the keywords for specifying the choice-type dependent variable:

- dependent <varname>
  <choice| ranking | bestworst | rating>

<joint> [missing] [unequal]
<rrm>
for bestworst: <anchoritem=varname>;

The four choice types are (first) choice, (sequential) ranking, (sequential) best-worst choice, and rating. Note that the modeling of ratings is very similar to the modeling or ordinal dependent variables. Moreover, it is important to note that in models with multiple dependent variables, the choice-type dependent variable should be the last one in the list of dependent variables. A LG Syntax model can contain only one choice-type dependent variable.

The option 'joint' makes it possible to model multiple choices from the same choice set as a single joint 'first' choice, a choice format which is sometimes referred to as pick-k-out-of-M. For ranking and best-worst models, it implies that these are no longer modelled assuming a sequential choice process, but that instead assuming a joint ranking or a joint selection of the best and worst alternatives. The 'missing' option in 'joint' can be used to change the treatment of situations in which a person selects less alternatives from a choice set than instructed. By default, the joint choice model is applied to the *selected* alternatives, but when using the 'missing' option, the missing choices are treated as missing values. By default the choice parameters are assumed to be the same across the multiple choices from a choice set (say between the first and second choice in a ranking, or between the best and worst choice). The 'unequal' option for joint choice models overrides this setting and indicates that each of the multiple choices has its own set of choice parameters.

The 'rrm' option is used to indicate that the choices should not be modelled using a standard random utility model (rum), but instead using a random regret model (Chorus, 2010, 2012).

Best-worst (or MaxDiff) experiments are sometimes supplemented with questions regarding the relevance of the alternatives for the respondent (Orme, 2009; Lattery, 2010). The best-worst choice task is followed with a question asking whether the alternatives 1) **all** are important, 2) **some** are important, some are not, or 3) **none** of these are important. The corresponding variable with response categories 1, 2, and 3, which is referred to as an anchor item, can be specified in best-worst models using the option 'anchoritem=<varname>'. To make this work with the 1-file data format, the choice sets should be expanded with the none alternative as the last option. With the 3-file data format, the alternatives file should contain the none alternative, and each choice set should have this alternative as the last option.

The specification of the attributes is very similar to the specification of independent variables:

- attributes <varname | (varlist)>
  <nominal | **numeric** | bspline [<degree=%d>] [knots=%d] [knots=(knot locations)]>
  <price>
  <coding= effects | first | last | %d>
  <score=(%f, %f,…, %f)>,
  next attribute, … ;

In addition to the attributes in the response file (1-file format) or the alternatives file (3-file format), the attribute '_Constants_' can be included in the list of attributes. It is used in the regression equation for the choice variable to specify the alternative-specific constants.

The keyword 'price' is used to indicate which of the attributes is the price variable. This information is needed for the computation of the willingness-to-pay (WTP) parameters.

4.4.4 CHOICE EQUATIONS

Choice equations look very much like other regression equations. The only options specific for choice models are the special effects '~rum' and '~rrm'. The former is used to indicate that a particular term in the regression equation should be modelled as a 'rum' term when the 'rrm' option was used in the definition of the dependent variable, and the latter is used to indicate the term concerned should be modelled as 'rrm' when the specified model is not an 'rrm' model.

4.5 SELECTING A KEYWORD, VARIABLE, FILE, OR MODEL

As indicated above, LatentGOLD 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 curser 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 LatentGOLD 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:

```
Item1 | Cluster;
Item2 | Cluster;
Item3 | Cluster;
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

```
Cluster <- 1;
Item1 - Item4 <- 1 | Cluster;
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:

```
Cluster <- 1;
Item1 <- (a) 1 | Cluster;
Item2 <- (a) 1 | Cluster;
Item3 <- (b) 1 | Cluster;
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:

```
Cluster <- 1;
Item1 <- (a) 1 | Cluster;
Item2 <- (b) 1 | Cluster;
Item3 <- (c) 1 | Cluster;
Item4 <- (d) 1 | Cluster;
b = a;
d = c;
```

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

```
b[1,] = a[1,];
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 item1 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:

```
Item1 <-> Item2;
```

A *class-specific* direct effect is defined as follows:

```
Item1 <-> Item2 | 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) Item1 <-> Item2 | 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:

```
Cluster <- 1 + sex + age;
```

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

```
Cluster <- 1 + sex + age + sex age;
```

or equivalently as

```
Cluster <- 1 + sex + age + sex * age;
```

A direct effect of a covariate on an indicator is specified as follows

```
Item3 <- 1 + Cluster + sex;
```

and an effect that varies across clusters as

```
Item3 <- 1 + Cluster + sex | 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 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:

```
Cluster <- 1 | sex;
```

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

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

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

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

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

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',

```
DFactor <- 1;
Item1 <- 1 + DFactor;
Item2 <- 1 + DFactor;
Item3 <- 1 + DFactor;
Item4 <- 1 + DFactor;
Item5 <- 1 + DFactor;
```

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:

```
DFactor1 <- 1;
DFactor2 <- 1;
Item1 <- 1 + DFactor1 + DFactor2;
Item2 <- 1 + DFactor1 + DFactor2;
Item3 <- 1 + DFactor1 + DFactor2;
Item4 <- 1 + DFactor1 + DFactor2;
Item5 <- 1 + DFactor1 + DFactor2;
```

or more compactly (but perhaps less readable) as

```
DFactor1 – DFactor2 <- 1;
Item1 - Item5 <- 1 + DFactor1 + DFactor2;
```

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:

```
DFactor1 <- 1;
DFactor2 <- 1;
DFactor1 <-> DFactor2;
Item1 <- 1 + DFactor1;
Item2 <- 1 + DFactor1;
Item3 <- 1 + DFactor2;
Item4 <- 1 + DFactor2;
Item5 <- 1 + DFactor1 + DFactor2;
```

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:

```
DFactor1 <- 1;
DFactor2 <- 1 + DFactor1;
Item1 <- 1 + DFactor1;
Item2 <- 1 + DFactor1;
Item3 <- 1 + DFactor2;
Item4 <- 1 + DFactor2;
Item5 <- 1 + DFactor1;
```

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:

```
DFactor1 <- 1 + sex + educ + age + sex educ;
DFactor2 <- 1 + DFactor1 + educ + age;
Item1 <- 1 + DFactor1;
Item2 <- 1 + DFactor1;
Item3 <- 1 + DFactor2;
Item4 <- 1 + DFactor2;
Item5 <- 1 + DFactor1;
```

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:

```
Factor0 <- 1;
Factor1 <- 1;
Factor0;
Factor1;
Factor0 <-> Factor1;
Item1 <- (1) Factor0 + (0) Factor1;
Item2 <- (1) Factor0 + (1) Factor1;
Item3 <- (1) Factor0 + (a) Factor1;
Item4 <- (1) Factor0 + (b) Factor1;
```

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

```
Factor0 <- 1;
Factor1 <- 1;
(1) Factor0;
(1) Factor1;
Item1 <- (a) Factor0 + (b) Factor1;
Item2 <- (a) Factor0 + (c) Factor1;
Item3 <- (a) Factor0 + (d) Factor1;
Item4 <- (a) Factor0 + (e) Factor1;
```

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:

```
Factor0 <- 1;
Factor1 <- 1;
Factor0;
Factor1;
Factor0 <-> Factor1;
Item1 <- (1) Factor0 + (1) Factor1 Time1;
Item2 <- (1) Factor0 + (1) Factor1 Time2;
Item3 <- (1) Factor0 + (1) Factor1 Time3;
```

```
Item4 <- (1) Factor0 + (1) Factor1 Time4;
```

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:

```
Factor0;
Factor1;
Factor0 <-> Factor1;
Item1 <- (a) 1 + (b) Time1 + (1) Factor0 + (1) Factor1 Time1;
Item2 <- (a) 1 + (b) Time2 + (1) Factor0 + (1) Factor1 Time2;
Item3 <- (a) 1 + (b) Time3 + (1) Factor0 + (1) Factor1 Time3;
Item4 <- (a) 1 + (b) Time4 + (1) Factor0 + (1) Factor1 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:

```
Cluster <- 1;
Factor0 <- 1 | Cluster;
Factor1 <- 1 | 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,

```
Item1 <- (a) 1 | Cluster + (b) Time1 | Cluster + (1) Factor0 + (1) Factor1 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:

```
Factor0 <- 1;
Factor1 <- 1;
Factor0;
Factor1;
Factor0 <-> Factor1;
Item <- (1) Factor0 + (1) Factor1 Time;
```

or as

```
Factor0;
Factor1;
Factor0 <-> Factor1;
Item <- 1 + Time + (1) Factor0 + (1) Factor1 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 LatentGOLD 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:

```
options
   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,

```
variables
  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:

```
MClass[=0] <- 1;
MClass <- 1 + MClass[-1];
Response <- 1 + MClass;
```

or using conditional effects as

```
MClass[=0] <- 1;
MClass <- 1 | MClass[-1];
Response <- 1 | MClass;
```

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

```
MClass[=0] <- 1;
MClass <- (~tra) 1 | MClass[-1];
Response <- (~err) 1 | MClass;
```

(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:

```
Response <- (b~err) 1 | 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,

```
MClass[=0] <- 1;
MClass <- (~tra) 1 | MClass[-1] time;
Response <- (~err) 1 | MClass;
```

or

```
MClass[=0] <- 1;
MClass <- (~tra) 1 | MClass[-1] + (~tra) time | MClass[-1];
Response <- (~err) 1 | 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:

```
MClass[=0] <- 1 | sex;
MClass <- (~tra) 1 | MClass[-1] sex + (~tra) time | MClass[-1] sex;
Response <- (~err) 1 | MClass sex;
```

Eliminating sex from one these equations would imply that one of the three types of model probabilities – initial, transition, or 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:

```
MClass[=0] <- 1 + sex;
MClass <- (~tra) 1 | MClass[-1] + (~tra) time | MClass[-1] + (~tra) sex | MClass[-1];
Response <- (~err) 1 | MClass;
```

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:

```
Class <- 1;
MClass[=0] <- 1 | Class;
MClass <- (~tra) 1 | MClass[-1] Class;
Response <- (~err) 1 | MClass;
```

or

```
Class <- 1;
MClass[=0] <- 1 + Class;
MClass <- (~tra) 1 | MClass[-1] + (~tra) Class | MClass[-1];
```

```
Response <- (~err) 1 | MClass;
```

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

```
MClass <- (~tra) 1 | MClass[-1] + (c~tra) Class | MClass[-1];
c[,1] = -100;
```

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-item4 <- 1 + Cluster;
```

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

```
(1) GCFactor;
Cluster <- 1 + GCFactor;
Item1-item4 <- 1 + Cluster;
```

5.6.2 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-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. Bias-adjusted step-three modeling
4. Regression models with cell weights (~wei option)
5. Modeling effects of independent variables using B-splines
6. Continuous-time Markov models
7. The knownclass indicator
8. Log-linear scale factor models for categorical dependent variables
9. Exploratory factor analysis with rotation to simple structure and agreement
10. Using a dynamic latent variable to define an additional level in a multilevel LC model
11. Syntax extensions for choice models

6.1 ALTERNATIVE REGRESSION MODELS FOR DICHOTOMOUS AND ORDINAL DEPENDENT VARIABLES

In the LatentGOLD 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 cumulative loglinear model (log)
6. the cumulative linear model (lin)
7. the continuation-ratio or sequential logit model (seqlogit1), and
8. 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, 7, and 8), probit (2), log-log (3 and 4), loglinear (5), and linear (6) models, respectively. Each of the six cumulative models uses a cumulative link function transforming the probability of responding in a particular category or higher into the linear predictor.

Note that one has to be cautious with the linear and log-linear models since the identity link (the linear model) does not guarantee that probabilities stay in the 0-1 range and the log link does not guarantee that probabilities are smaller than 1. There is, therefore, no guarantee that the algorithms implemented in LatentGOLD will

converge. On the other hand, sometimes it is convenient to define constraints on probabilities in linear or log-linear form, which is why we implemented these models.

The Technical Guide provides details on these binary and ordinal regression 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 additional 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. The Dirichlet regression model is a multivariate generalization of the beta regression model for a set of non-negative variables summing to 1.
- 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

```
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 BIAS-ADJUSTED STEP-THREE MODELING

LatentGOLD Basic contains a 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, 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:

```
options
    step3 proportional ml;
    output parameters standarderrors estimatedvalues;
variables
   independent z1, z2, z3;
   latent Cluster nominal posterior=(Cluster#1 Cluster#2 Cluster#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=(Cluster#1 Cluster#2 Cluster#3) to define the names of the variables containing the posterior class 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:

```
options
   step3 modal bch;
   output parameters standarderrors estimatedvalues profile;
variables
   dependent y1 ordinal, y2 nominal, y3 continuous;
   latent Cluster nominal posterior=(Cluster#1 Cluster#2 Cluster#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 class 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, 2016).

Let us return to the Step3-Covariate example and suppose z1 is an independent variable that was also used in the step-1 model, and which, moreover, turned out to have a direct effect on one or more of the indicators. This implies that we are in the situation that there is differential item function (DIF) or measurement non-invariance

(see, Vermunt and Magidson, 2021). This is something that should be indicated as follows in the step-3 Syntax model:

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

As can be seen, we add 'dif=z1' to the latent variable definition. If both z1 and z2 are DIF variables, the specification becomes 'dif=(z1 z2)'.

As an alternative to the ml or bch approach, one may perform stepwise LC modeling using the Bakk-Kuha approach. This requires writing the log-densities to an outfile, and using the 'logdensity' instead of the 'posterior' keyword in the latent variable definition. That is:

```
options
    step3 proportional ml;
    output parameters standarderrors estimatedvalues;
variables
   independent z1, z2, z3;
   latent Cluster nominal logdensity=(logdensity1 logdensity2 logdensity3);
equations
   Cluster <- 1 + z1 + z2 + z3;
```

In addition to the above examples, we will illustrate several more general step-3 Syntax models that are not available in the Step3 GUI. The first one concerns a model containing both independent and dependent variables. That is,

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

As can be seen, we use z1, z2, and z3 as control variables in the regression of y on Cluster. Note that we use the 'simultaneous' option to indicate that the regression models for the latent variable Cluster and the dependent variable y should be estimated simultaneously instead of one by one.

Another example is a (path) model with 2 latent variables:

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

An example of a step-three latent Markov model with predictors affecting the transition probabilities is:

```
options
   step3 modal ml;
   output parameters standarderrors estimatedvalues;
variables
   caseid id;
   independent z1, z2, z3;
   latent State nominal dynamic posterior=(Class#1 Class#2);
equations
   State[=0] <- 1;
   State <- (~tra) 1 | State[-1] + (~tra) z1 | State[-1] + (~tra) z2 | State[-1] + (~tra) z3 | State[-1];
```

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

```
options
   step3 modal ml;
   output parameters standarderrors estimatedvalues;
variables
   groupid teamid;
   independent z1, z2, z3;
   latent U continuous, Class nominal posterior=(Class#1 Class#2);
equations
   U;
   Class <- 1 + (1) U + z1 + z2 + z3;
```

Note that in models for data sets with more complex dependence structures, only the modal-ml, the modal-none, and the Bakk-Kuha approaches can be used. This applies to latent Markov models and multilevel LC models.

6.4 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 (transitions 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 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 '{…}'.

6.5 MODELING THE EFFECT OF INDEPENDENT VARIABLES USING B-SPLINES

B-spline (basis spline) smoothers allow for flexible modeling of the relationship between quantitative predictors and outcome variables (de Boor, 1978). In LG Syntax, this is implemented as an option for independent variables, and in choice models also for attributes. The B-spline option is especially useful for the modeling of the time dependence in (latent class) growth models (see Francis, Elliot, and Weldon, 2016).

This is sample Syntax illustrating the bspline option in a LC growth model:

```
variables
   latent Class nominal 3;
   independent time bspline degree=3 knots=2;
equations
   Class <- 1;
   y <- 1 | Class + time | Class;
```

As can be seen, 'bspline' is used instead of the 'nominal' or 'numeric' setting for the independent variable time. The degree and the number of inner knots of the basis splines are set to 3 and 2, respectively.

It is also possible to specify the knots' locations instead of the number of knots. An example is:

```
independent time bspline degree=3 knots=(0,5,10,20);
```

where 0 and 20 are the values of the two outer knots and 5 and 10 the two inner knots.

The B-spline option can be combined with the "+" and "-" monotonicity restrictions on parameters. For example, in the above LC growth model one may restrict the time effect is monotone (either increasing or decreasing) for some or all latent classes. An example is:

```
variables
   latent Class nominal 3;
   independent time bspline degree=3 knots=2;
equations
   Class <- 1;
   y <- 1 | Class + (b) time | Class;
   b[1] = +;
   b[2] = -;
```

In this model, the time effect is specified to be increasing for Class 1, decreasing for Class 2, and unrestricted for Class 3.

6.6 CONTINUOUS-TIME MARKOV MODELS

LG-Syntax contains an 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 LatentGOLD 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 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 either 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 dtime;
   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);

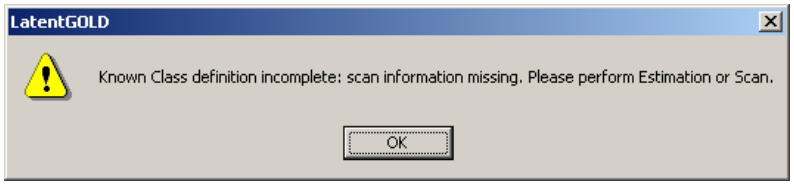
Here, the assignments are as explained above except that TRUE is a string variable with codes 'a', 'b', and 'c'.

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 LOG-LINEAR SCALE FACTOR MODELS FOR DEPENDENT VARIABLES

A special modeling feature is the possibility to include a scale factor in regression models for categorical responses (nominal and the various types of ordinal and choice dependent variables), and for continuous responses and counts. A scale factor is a term by which all parameters in the regression model are multiplied, thus modeling proportionality of parameter values across 'groups'. For categorical responses, 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 a scale model implies that the linear term in the regression model for the 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 capturing its dependence on latent and/or independent variables. The Technical Guide provides more information of these log-linear scale factor models.

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

```
variables
   dependent y cumlogit;
   independent time, female;
equations
```

```
y <- 1 + time + female;
y <<- female;
```

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

Subsection 6.11.3 presents a choice modeling application in which the log-linear scale factor model contains a scale latent class variable (referred to as sClass). There, the scale model is used to better distinguish heterogeneity in preferences from heterogeneity in uncertainty about preferences (Magidson and Vermunt, 2007).

6.9 EXPLORATORY FACTOR ANALYSIS WITH ROTATION TO SIMPLE-STRUCTURE AND AGREEMENT

The LG-Syntax module implements factor rotation options which can be used to obtain interpretable and identified exploratory factor analysis (EFA) solutions. The rotation criteria implemented to achieve a simple structure are varimax, oblimin, geomin, and target rotation. This is sample Syntax for a simple EFA model with 2 correlated factors:

```
options
   <other options>
   rotation oblimin;
variables
   latent F1 continuous, F2 continuous;
   dependent (Y1-Y10) continuous;
equations
   (1) F1;
   (1) F2;
   F1 <-> F2;
   Y1-Y10 <- 1 + F1 + F2;
```

As can be seen, the model contains all terms, including the factor covariance. The factor variances are fixed to 1 for identification. The rotation criterion used is ‘oblimin’.

An example of a multiple-group EFA with 2 correlated factors and with country being the grouping variable is:

```
options
   <other options>
   rotation oblimin procrustes=0.5;
variables
   independent country nominal;
   latent F1 continuous, F2 continuous;
   dependent (Y1-Y10) continuous;
equations
   F1 | Country;
   F2 | Country;
   F1 <-> F2 | Country;
   Y1 – Y10 <-  1 | Country + F1 | Country + F2 | Country;
```

As can be seen, the model contains all terms, including the factor covariance, and each of these is allowed to be country specific. Whereas the factor variances are specified to be free, they will be restricted to have a mean of 1 across countries. The specified rotation criterion is a combination of 'oblimin' and 'procrustes', each with a weight of 0.5. Typically, this will yield a good balance between rotation to simple structure within countries and to agreement between countries (De Roover and Vermunt, 2019).

Similarly, one may define a mixture EFA model:

```
options
   <other options>
   rotation oblimin procrustes=0.5;
variables
   latent Class nominal 2, F1 continuous, F2 continuous;
   dependent (Y1-Y10) continuous;
equations
   F1 | Class;
   F2 | Class;
   F1 <-> F2 | Class;
   Y1 – Y10 <-  1 | Class + F1 | Class + F2 | Class;
```

6.10 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 yl, 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;
```

6.11 SYNTAX EXTENSION FOR CHOICE MODELS

The LG-Syntax™ module extends the capabilities of LatentGOLD Choice in several important ways, such as:

- Inclusion of a log-linear scale factor model with scale classes in addition to preference classes. Choice responses elicited from persons with lower scale factors exhibit higher error variance, indicative of higher amounts of uncertainty. Inclusion of scale factors in LC choice models provides better separation and clearer interpretation of the latent class segments (Magidson and Vermunt, 2007).
- Estimation of *fused* models with a single common LC segmentation. Data from a discrete-choice design may be analyzed in conjunction with one or more ratings or other (non-choice) dependent variables to obtain a common LC segmentation that applies to both the choice and non-choice dependent variables.
- Inclusion of multiple latent variables, for example, for the estimation of fused models to investigate the relationship between a single LC segmentation for the discrete choice data and a second for explaining heterogeneity with respect to the other dependent variables.
- Dynamic (time-varying) latent variables, flexible parameter restrictions, and options for Monte Carlo simulation, validation and hold out, saving a model with parameter estimates for rapid re-estimation and scoring of other data files, submitting runs in batch mode, etc.

The remainder of this section provides examples of the LG Syntax for a simple choice model and for choice models using an extended data structure to accommodate the estimation of fused models.

6.11.1 EXAMPLE 1: A BASIC CHOICE MODEL

To see what the LG-Syntax looks like in the case of a choice model, one can simply open a Syntax choice example from the Examples Menu, or open a GUI choice example (from the Examples Menu or a previously saved .lgf file) and from the Model Menu select 'Generate Syntax' to view the syntax statements in the Syntax Editor. Sample Syntax for a simple LC choice model using a 1-file format is:

```
options
   <options>
choice = 1;
variables
   caseid id;
   choicesetid set ;
   dependent ch choice;
   attribute brand, cap, price, filter, therm;
   latent Class nominal 3 ;
equations
   Class <- 1 ;
   ch <- brand | Class + cap | Class + price | Class + filter | Class + therm | Class;
```

Note that the statement ‘choice = 1;’ refers to the 1-file format. The only change required with a 3-file structure is to replace the statement ‘choice = 1’ with the following statements which refer to the 3-file structure and provide the file names for the alternatives and sets files:

```
choice = 3
   alternatives 'coffee_alternatives.say'
   id=altid
   choicesets 'coffee_sets.say'
   id=setid ;
```

6.11.2 EXAMPLE 2: DATA STRUCTURE FOR CHOICE PLUS OTHER DEPENDENT VARIABLES

When a choice model is combined with other types of dependent variables, the data file contains one or more additional records with missing values in the choice fields (choice set id #, choice, etc.) to accommodate the other dependent variables. Similarly, the records pertaining to the choice information contain missing values in the fields that pertain to the other dependent variables, including fields that may define independent variables that pertain only to one or more of these other dependent variables.

For example, as shown in Figure 6-1, assume respondent #1 (caseid = 1) provides responses to 4 choice sets, the choices being recorded in dependent variable `dep1', and responses are also provided to 2 other dependent variables -- ordinal dependent variable `dep2', and continuous dependent variable `dep3'. In addition, suppose that there are 3 independent variables. The first, `indep1', is used as a covariate (i.e., as a predictor of the latent classes), and the others, `indep2' and `indep3', as predictors of the dependent variables `dep2' and `dep3'.

caseid	setid	indep1	indep2	indep3	dep1	dep2	dep3
1	1	1	.	.	2	.	.
1	2	1	.	.	1	.	.
1	3	1	.	.	1	.	.
1	4	1	.	.	3	.	.
1	.	1	2	2	.	4	3.24

Figure 6-1: Response file for the 3-file data structure

Caseid #1 selects alternative #2 to set 1, alt. #1 to sets 2 and 3, and alt. #3 to set 4. The last record with a missing value on the setid and dep1 contains additional responses (dep2 and dep3) to non-choice related survey questions.

The corresponding 1-file structure shown in Figure 6-2 contains not only the responses but also information regarding alternatives, attributes, and choice sets. To accomplish this, separate rows are required for each alternative in each choice set. Note also that unlike the 3-file format, the choice response variable ‘dep1’ is now a dummy variable coded ‘1’ or ‘0’, where ‘1’ flags the selected alternative.

caseid	setid	att1	att2	att3	indep1	indep2	indep3	dep1	dep2	dep3
1	1	1	1	2	1	.	.	0	.	.
1	1	2	2	1	1	.	.	1	.	.
1	1	3	2	2	1	.	.	0	.	.
1	1	1	1	1	1	.	.	0	.	.
1	2	3	2	2	1	.	.	1	.	.
1	2	3	3	3	1	.	.	0	.	.
1	2	2	1	2	1	.	.	0	.	.
1	2	3	2	3	1	.	.	0	.	.
1	3	3	3	3	1	.	.	1	.	.
1	3	2	1	2	1	.	.	0	.	.
1	3	3	3	2	1	.	.	0	.	.
1	3	2	3	1	1	.	.	0	.	.
1	4	3	2	4	1	.	.	0	.	.
1	4	2	3	4	1	.	.	0	.	.
1	4	3	2	4	1	.	.	1	.	.
1	4	2	3	4	1	.	.	0	.	.
1	.	.	.	.	1	2	2	.	4	3.24

Figure 6-2: Response file for the 1-file data structure

Irrespective of whether the 1- or 3-file setup is used, the variables and equations part of the model specified for this example could be as follows:

```
variables
   caseid caseid;
   choicesetid setid ;
   dependent dep2, dep3 continuous, dep1 choice;
   independent indep1, indep2, indep3;
   attribute att1, att2, att3;
   latent Class nominal 2;
equations
   Class <- 1 + indep1;
   dep2 <- 1 | Class + indep2 | Class + indep3 | Class;
   dep3 <- 1 | Class + indep2 | Class + indep3 | Class;
   dep1 <- att1 I Class + att2 | Class + att3 | Class;
```

6.11.3 EXAMPLE 3: DATA FUSION EXAMPLE WITH A SCALE MODEL AND A CONTINUOUS FACTOR

A more advanced example application concerns a study in which 305 lab managers responded to MaxDiff (best-worst) questions involving the importance of 36 attributes posed in sets of 5 at a time, followed by records pertaining to ratings (5-point scale) for each of these 36 attributes. This application is described in more detail in Magidson, Thomas, and Vermunt (2009) and Vermunt (2013).

This data fusion example uses the extended 3-file structure discussed above. The variables and equations sections of the Syntax have the following form:

```
variables
   caseid ID;
   choicesetid setid;
   dependent RATING cumlogit, CHOICE bestworst;
   independent Feature nominal;
   attribute attFeature nominal;
   latent Class nominal 5, sClass nominal 2, u continuous;
equations
   Class <- 1;
   sClass <- 1;
   CHOICE <<- (s) 1 | sClass;              // scale model for choice
   RATING <<- 1 | sClass;                  // scale model for rating
   (1) u;                                  // variance of continuous latent variable fixed to 1
   CHOICE <- (b1) attFeature | Class;      // model for choice
   RATING <- 1 + Class + u + (b2) Feature | Class; // model for rating
    s[1]=0;                                // restriction needed for identification
   b2=b1;                                  // bestworst and rating parameters are equated
```

As can be seen, a second nominal latent variable (sClass) is included in the model to account for differences in scale between the choice and the rating parameters and between individuals. Moreover, a continuous latent variable (u) is added to account for the fact that individuals differ in the way they respond to rating questions.

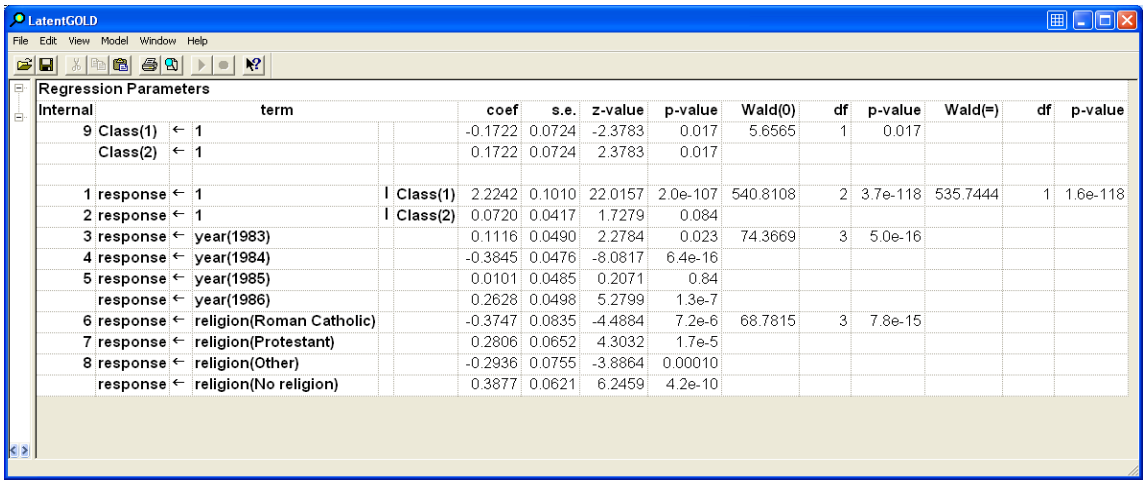
7 DETAILS ON VARIOUS SYNTAX TOOLS AND OUTPUT OPTIONS

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. Asymptotic power computation for likelihood-ratio tests
8. Score tests and EPCs
9. Alternative complex sampling options
10. Identification checking
11. Validation and hold-out options
12. 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 7-1 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.



Internal	term			coef	s.e.	z-value	p-value	Wald(0)	df	p-value	Wald(=)	df	p-value
9	Class(1) ← 1			-0.1722	0.0724	-2.3783	0.017	5.6565	1	0.017			
	Class(2) ← 1			0.1722	0.0724	2.3783	0.017						
1	response ← 1	|	Class(1)	2.2242	0.1010	22.0157	2.0e-107	540.8108	2	3.7e-118	535.7444	1	1.6e-118
2	response ← 1	|	Class(2)	0.0720	0.0417	1.7279	0.084						
3	response ← year(1983)			0.1116	0.0490	2.2784	0.023	74.3669	3	5.0e-16			
4	response ← year(1984)			-0.3845	0.0476	-8.0817	6.4e-16						
5	response ← year(1985)			0.0101	0.0485	0.2071	0.84						
	response ← year(1986)			0.2628	0.0498	5.2799	1.3e-7						
6	response ← religion(Roman Catholic)			-0.3747	0.0835	-4.4884	7.2e-6	68.7815	3	7.8e-15			
7	response ← religion(Protestant)			0.2806	0.0652	4.3032	1.7e-5						
8	response ← religion(Other)			-0.2936	0.0755	-3.8864	0.00010						
	response ← religion(No religion)			0.3877	0.0621	6.2459	4.2e-10						

Figure 7-1: Parameters with 'Internal' column.

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:

```
equations
```

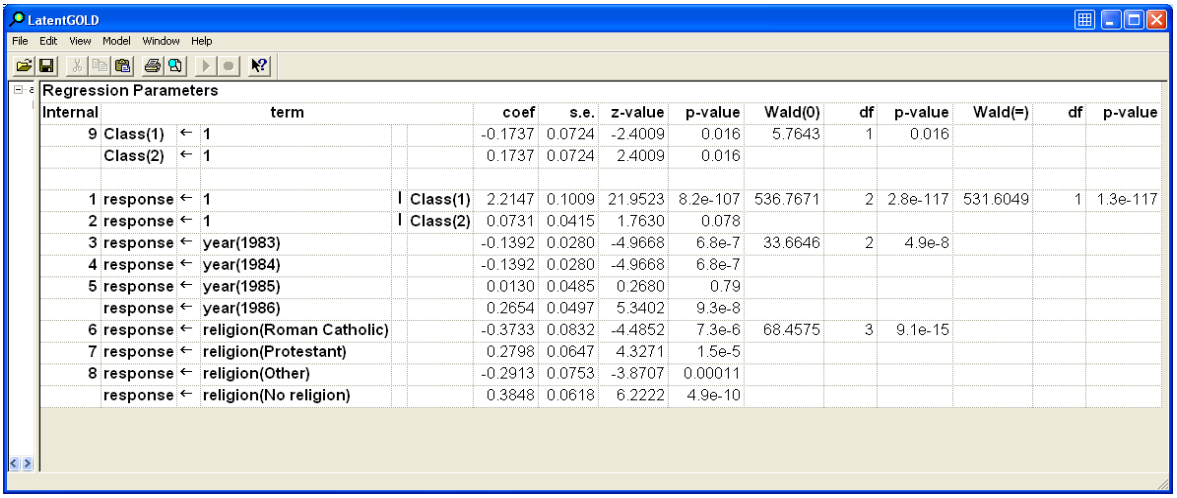
```
response <- 1 | Class + year + religion ;
Class <- 1;
{2.224168744663137
0.07198244434239946
0.1115934817516651
-0.3844805646738486
0.01005171849917819
-0.3747469959227842
0.2806252836538088
-0.2935911967842039
-0.1721545534454281}
```

The numbers between '{…}' are the saved parameter values. Comparing these numbers to those given in the 'Coef' column (see Figure 7-1) 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 than the reported parameters themselves. An example is provided in Figure 7-2, which shows the result of running the same model as above but now with a monotonic effect of 'year'.



Internal			term		coef	s.e.	z-value	p-value	Wald(0)	df	p-value	Wald(=)	df	p-value
9	Class(1)	←	1		-0.1737	0.0724	-2.4009	0.016	5.7643	1	0.016			
	Class(2)	←	1		0.1737	0.0724	2.4009	0.016						
1	response	←	1	Class(1)	2.2147	0.1009	21.9523	8.2e-107	536.7671	2	2.8e-117	531.6049	1	1.3e-117
2	response	←	1	Class(2)	0.0731	0.0415	1.7630	0.078						
3	response	←	year(1983)		-0.1392	0.0280	-4.9668	6.8e-7	33.6646	2	4.9e-8			
4	response	←	year(1984)		-0.1392	0.0280	-4.9668	6.8e-7						
5	response	←	year(1985)		0.0130	0.0485	0.2680	0.79						
	response	←	year(1986)		0.2654	0.0497	5.3402	9.3e-8						
6	response	←	religion(Roman Catholic)		-0.3733	0.0832	-4.4852	7.3e-6	68.4575	3	9.1e-15			
7	response	←	religion(Protestant)		0.2798	0.0647	4.3271	1.5e-5						
8	response	←	religion(Other)		-0.2913	0.0753	-3.8707	0.00011						
	response	←	religion(No religion)		0.3848	0.0618	6.2222	4.9e-10						

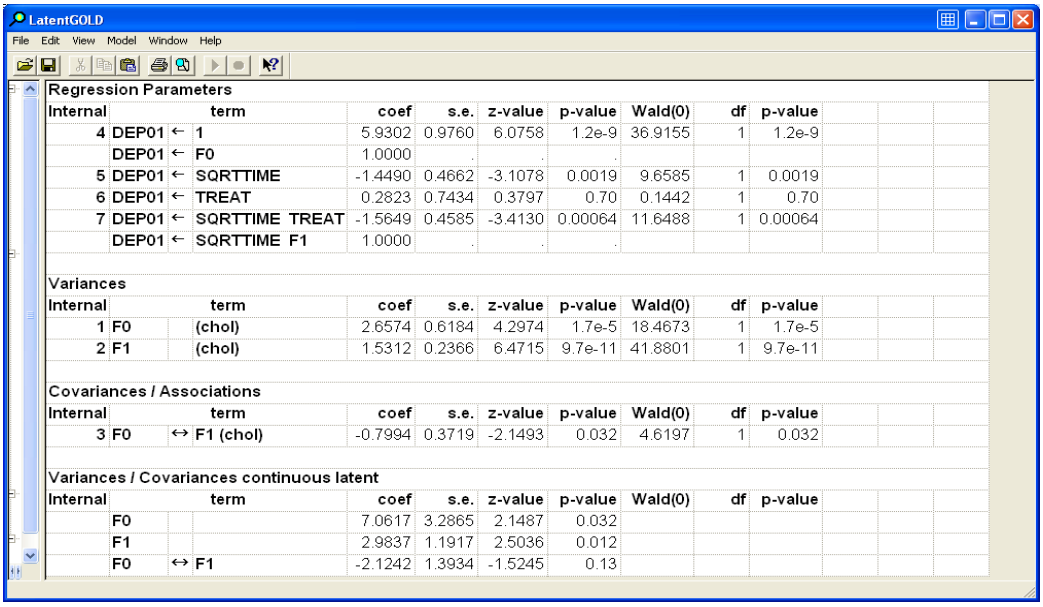
Figure 7-2: Parameters output with internal parameter numbers for a model with monotonicity constraints.

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
   0.2797925487051707
   -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.



LatentGOLD

File Edit View Model Window Help

Regression Parameters

Internal			term	coef	s.e.	z-value	p-value	Wald(0)	df	p-value
4	DEP01	←	1	5.9302	0.9760	6.0758	1.2e-9	36.9155	1	1.2e-9
	DEP01	←	F0	1.0000	.	.	.			
5	DEP01	←	SQRTTIME	-1.4490	0.4662	-3.1078	0.0019	9.6585	1	0.0019
6	DEP01	←	TREAT	0.2823	0.7434	0.3797	0.70	0.1442	1	0.70
7	DEP01	←	SQRTTIME TREAT	-1.5649	0.4585	-3.4130	0.00064	11.6488	1	0.00064
	DEP01	←	SQRTTIME F1	1.0000	.	.	.			

Variances

Internal			term	coef	s.e.	z-value	p-value	Wald(0)	df	p-value
1	F0		(chol)	2.6574	0.6184	4.2974	1.7e-5	18.4673	1	1.7e-5
2	F1		(chol)	1.5312	0.2366	6.4715	9.7e-11	41.8801	1	9.7e-11

Covariances / Associations

Internal			term	coef	s.e.	z-value	p-value	Wald(0)	df	p-value
3	F0	↔	F1 (chol)	-0.7994	0.3719	-2.1493	0.032	4.6197	1	0.032

Variances / Covariances continuous latent

Internal			term	coef	s.e.	z-value	p-value	Wald(0)	df	p-value
	F0			7.0617	3.2865	2.1487	0.032			
	F1			2.9837	1.1917	2.5036	0.012			
	F0	↔	F1	-2.1242	1.3934	-1.5245	0.13			

Figure 7-3: Cholesky parameters and Variances/Covariances of Continuous Latent Variables

Figure 7-3 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 7-3 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

7.0617	-2.1242
2.1242	2.9837

and that lower-diagonal Cholesky matrix and its transpose equal

2.6574	0
-0.7994	1.5312

The covariance matrix equals the Cholesky matrix times its transpose

2.6574	-0.7994
0	1.5312

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 LG 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 2.2). Assign a file name and select 'Save'.
3. Use a text editor to edit the 'lgs' file that was created.
    a. Change the name of the infile data file on line 4 to the name of the new file that you want to score.
    b. Since we don't want to estimate the model parameters, but instead keep the parameters fixed to their saved values, modify the settings referring to the maximum number of iterations to be performed to 0 (thus, emiterations=0 and nriterations=0 in algorithm and iterations=0 in startvalues).
    c. Optionally, you may remove irrelevant items from the list of output.
    d. Finally, after the output section, insert the keyword 'outfile', followed by the name of the output file to be created and by the keywords describing the output you wish to create -- 'classification', 'prediction', etc. as indicated on section 4.1.12. For example, assuming that you are using SPSS data 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.
4. From the File menu of LatentGOLD, 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.

It should be noted that in Cluster- and DFactor-like Syntax models, posterior classifications can be added to external files also without using LatentGOLD. That is, SPSS, R, or generic classification syntax can be generated with the WriteSPSSSyntax, WriteRSyntax, or WriteGenSyntax output options.

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=%d'.

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 emiterations=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 simulated 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
   49}
```

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

```
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 BOOTSTRAP, VUONG, AND ASYMPTOTIC LIKELIHOOD-RATIO TESTS

LatentGOLD Syntax implements the following likelihood-ratio tests:

- Bootstrap and Vuong tests for comparing models with different numbers of classes when running models for a range of classes.
- Bootstrap, Vuong, and asymptotic tests for comparing the current H1 model with a user-specified H0 model.

This is sample Syntax showing how to run models from 1 to 4 classes:

```
   options
```

```
    <specification of technical and output options>
variables
    dependent Y1, Y2, Y3, Y4, Y5;
    latent  Cluster nominal 1:4;
equations
    Cluster  <- 1;
    Y1 – Y5 <- 1 + Cluster;
```

As can be seen, the number of classes is specified as "1:4". When running this Syntax, one will automatically obtain Vuong tests for the comparison of models with 1 and 2, 2 and 3, and 3 and 4 classes. Lo, Mendell, and Rubin (2001) proposed using Vuong's robust likelihood-ratio test for this purpose, which is why it is often referred to as the VLMR or LMR test.

If, in addition, one wishes to obtain bootstrap p-values for the same set of model comparisons, one should add the keyword "LLdiff" to the montecarlo options; that is,

```
options
    <other options>
    montecarlo sets=16 replicates=500 LLdiff;
variables
    dependent Y1, Y2, Y3, Y4, Y5;
    latent  Cluster nominal 1:4;
equations
    Cluster  <- 1;
    Y1 – Y5 <- 1 + Cluster;
```

As can be seen, in addition to the LLdiff option for requesting the bootstrap likelihood-ratio test (BLRT), we use the option "sets=16" to indicate that 16 random start sets should be used for the bootstrap replicates.

Vuong, bootstrap, and asymptotic likelihood-ratio tests can be performed between any pair of models. For the Vuong and asymptotic test, this requires using the output option "Vuong='H0 model name'" and "LLdiff='H0 model name'", respectively. Sample Syntax showing how to request both types of test is:

```
options
    <other options>
    output parameters standarderrors LLdiff='restricted 3-class' Vuong='restricted 3-class';
variables
    dependent Y1, Y2, Y3, Y4, Y5;
    latent  Cluster nominal 3;
equations
    Cluster  <- 1;
    Y1 – Y5 <- 1 + Cluster;
```

In this example, we assume the two tests are used to compare an unrestricted 3-class model with a restricted 3-class model. Note that the Vuong test could also be used to compare the 3-class model with the 2-class model, which would be equivalent to running models with 2 and 3 classes using the "2:3" option for the number of classes.

Comparing an arbitrary pair of nested models using the BLRT is achieved with the LLdiff option in montecarlo, but now with a specification of the H0 model; that is,

```
options
   <other options>
   montecarlo sets=16 replicates=500 LLdiff='restricted 3-class';
variables
   dependent Y1, Y2, Y3, Y4, Y5;
   latent  Cluster nominal 3;
equations
   Cluster  <- 1;
   Y1 – Y5 <- 1 + Cluster;
```

Note that while the bootstrap and the asymptotic test require the H0 and H1 model to be nested models, versions of the Vuong test are available for comparing nested, overlapping, and non-nested models (see Merkle, You, and Preacher, 2016).

7.6 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 either parameter labels or (internal) parameter numbers. Note that each constraint is enclosed within braces '{…}'.

Examples are '{(1) a[1,1] = (2)}' and '{(1) a[1,1] + (-1) b[1,1] = (0)}', indicating that the parameter a[1,1] equals 2 and that parameters a[1,1] and b[1,1] are equal. These two restrictions can also be formulated in a more compact way using the fact that the constants (c1), (c2), etc. can be omitted (default is (1)), that it is allowed to use a minus instead of a plus sign, and that the '= (d)' can be omitted (default is '= (0)'). This yields '{a[1,1] = (2)}' and '{a[1,1] – b[1,1]}', respectively.

When using parameter numbers instead of parameter labels, the two example restrictions could be of the form '{6 = (2)}' and '{6 - 12}', where a[1,1] and b[1,1] from above are replaced by their parameter numbers 6 and 12.

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.

7.7 USER-DEFINED EXPRESSIONS

Often we are interest in functions of model parameters. Sometimes, getting their point estimates is enough, but in other situations we may also wish to quantify their uncertainty (their SEs) or use them in univariate z or multivariate Wald tests. Examples of applications in which this is useful include the computation of indirect effects, the computation of parameters with a different coding, and the computation of predicted values for certain values of the latent and/or independent variables.

LatentGOLD Syntax implements an option for specifying user-defined expressions or functions. These expressions may contain parameter labels, names of other expressions, numbers, operators (+, -, /, *, and ^), and mathematical functions like sqrt(), log(), and exp(). These expressions can either be defined at the end of the equations section, together with the parameter constraints, or in a separate file defined in the output section using expressions=<filename>.

The general form of an expression appearing in the equations section is:

```
%<name> = <expression>;
```

The name of an expression should be unique in the sense that it was not used as variable name or parameter label in the model specification, and when used in the equations section it should start with the symbol '%'. More specifically, if the program encounters a line starting with a % in the equations section, it recognizes it is an expression. Within an expression, it is possible to use the name of other expressions (without the symbol '%').

This is sample Syntax of a model with an indirect effect of a 4-class latent variable Cluster on a continuous outcome Y via a continuous mediating variable X (for which we have two copies in the data file, namely Xdep and Xindep):

```
equations
  Cluster <- 1;
  Xdep <- 1 + (a) Cluster;
  Y <- 1 + (b) Xindep;
  Xdep;
  Y;
  %c1 = a[1,1]*b[1,1];          // indirect effect for Cluster 1
  %c2 = a[1,2]* b[1,1];         // indirect effect for Cluster 2
  %c3 = a[1,2]* b[1,1];         // indirect effect for Cluster 3
  %c4 = - c1 - c2 - c3;         // indirect effect for Cluster 4
```

As can be seen, we compute the products of the 3 parameters for the effect of Cluster on X and the effect of X on Y. Because of the effect coding scheme, the indirect effect for Cluster 4 equals the negative of the sum of the indirect effects for Cluster 1, 2, and 3. Note that the above model could be a step-3 model using latent classes obtained from a step-1 analysis.

LG-Syntax allows specifying the above expressions in a more compact manner using the vector and sum options. That is,

```
%c[1:3] = a[1,1:3]*b[1,1];    // indirect effects for Cluster 1-3
%c[4] = c[+];                 // indirect effect for Cluster 4
```

Nested within the Parameters output, the program will report the values of the user expressions, as well as their SEs and corresponding z tests. Note that by default the tests are based on asymptotic standard errors, but it is possible to use any of the other standard error estimators implemented in LatentGOLD.

The expressions can also be used in the definition of user-defined Wald tests. For the same example, the file with the user-defined Wald tests could have the following form:

```
// pairwise comparisons of indirect effects across Clusters
{c1-c2};
{c1-c3};
{c1-c4};
{c2-c3};
{c2-c4};
{c3-c4};
// overall test of indirect of Cluster on y
{c1}{c2}{c3};
```

This yields Wald tests for the contrasts of the indirect effects between pairs of classes, and for the overall indirect effect consisting of three (non-redundant) terms.

7.8 POWER COMPUTATION FOR WALD TESTS

In LG-Syntax it possible to perform power computations for the Wald tests, including the user-defined Wald test discussed above (Gudicha, Tekle, and Vermunt, 2016). 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, including all its parameter values;
- 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.

7.9 ASYMPTOTIC POWER COMPUTATION FOR LIKELIHOOD-RATIO TESTS

Using LG-Syntax, one can perform asymptotic power computations for the likelihood-ratio tests comparing nested model (Gudicha, Schmittmann, and Vermunt, 2016). It works as follows:

- Open either an 'exemplary' data set or a large simulated data set under the H1 model;
- Specify the assumed H1 population model;
- Specify the restricted H0 model;
- Include the command 'LLdiff=H0model' and 'LLdiffPower=<number>' in the list of output option of the H1 model;

- Run the H1 model.

When the specified number in LLdiffPower 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.

An exemplary data set can be created using the 'writeexemplarydata' output option, while a large data set can be simulated using the outfile option 'simulation'. Note that both require specifying the assumed H1 population model, including all its parameter values.

7.10 SCORE TESTS AND EPCS

LG-Syntax implements a 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, 2016). By EPC(self) we refer to the expected changes in the parameters of the restricted set itself after removing the restriction concerned.
- An output section nested within Parameters containing the EPC(other) values for all free parameters (Oberski, 2014; Oberski and Vermunt, 2013, Oberski, Vermunt, and Moors, 2015). 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).

If one wishes to test the significance of specific sets of (restricted) parameters simultaneously, one should use 'ScoreTest=<filename>', where the file lists the parameter names of each set one would like to test, and where sets are separated by a semi-colon ';'. An example for the content of the file is:

```
ab ac ad ae bc bd be cd ce de;      // all direct effects
ab ac ad ae;                        // all direct effects involving variable a
ab bc bd be;                        // all direct effects involving variable b
ac bc cd ce;                        // etc.
ad bd cd de;
ae be ce de;
```

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 (2016).

7.11 ALTERNATIVE STANDARD ERROR ESTIMATORS

The options for standard error estimation with complex sampling designs are expanded in LG-Syntax beyond the capabilities of LatentGOLD 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.

In LG-Syntax 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.

Finally, when using the SimulateParameters option, parameter sets are sampled directly from the MVN distribution defined by their point estimates and their estimated asymptotic covariance matrix. Relevant functions of parameters are computed with each sample and their standard deviations across samples serve as SE estimates. Those functions can be the LatentGOLD output sections like Profile and EstimatedValues, but can also be user-defined expressions as described above.

7.12 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 for 2 to 5 nominal dependent variables

Number of Latent Classes	Number of Dependent Variables	Number of Categories Dependent Variable	Number of Nonidentified Parameters
2	2	any	2
2	>=3	any	0
3	2	both 2	5
3	2	one >=3	6
3	3	all 2	4
3	3	one >=3, two 2	3
3	3	two >=3	0
3	4	all 2	1
3	4	one >=3	0
3	>=5	any	0
4	2	both 2	8
4	2	one 3, one 2	10
4	2	both 3	11
4	2	one >=4	12
4	3	two or three 2	8
4	3	two >=3, one 2	6
4	3	all 3	2
4	3	two >=3, one >=4	0
4	4	all 2	4
4	4	one >=3	0
4	>=5	any	0
5	2	both 2	11
5	2	both 3	16
5	2	one >=5	20
5	3	all 2	12
5	3	all 3	8
5	3	all >=4	0
5	4	all 2	9
5	4	three >=3	0
5	4	one >=3 and one >=4	0
5	>=5	any	0

7.13 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). Statistics are, however, 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. Prediction statistics are, however, also reported for the hold-out replications, where for posterior and hblike 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.

7.14 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 this subsection, we describe the most important write options, which are:

- write/append='filename',
- WriteExemplaryData='filename',
- WriteParameters='filename',
- WriteHessian='filename',
- WriteGradient='filename',

- WriteBCH='filename',
- WriteClassificationErrors='filename', and
- WriteClassificationErrorsProportional='filename'
- WriteFactorErrors='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 appends 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, 2014). We used these options for our research of three-step LC analysis and during the implementation of the step-three modeling options in LatentGOLD. WriteFactorErrors='filename' does the same for continuous variables; that is, it yields the loadings and residual variances of the factors scores, as well as the information needed for the first- and second-order corrections of the step-three standard errors.

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', WriteRSyntax='filename', or WriteGenSyntax='filename', one can create an SPSS, R, 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

This tutorial introduces the use of the LG-Syntax module, an add-on to the Basic version of LatentGOLD. 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 LatentGOLD using the 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 Set

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. The name of the data set is **'conjoint.sav'**, which is located in the folder called **DemoData.**

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+).*

CONJOINT.SAV - SPSS Data Editor

File Edit View Data Transform Analyze Graphs Utilities AmosApps Window Help

1 : id 1

	id	sex	age	fashion	quality	price	rating
1	1	Male	25-39	Traditional	Low	Higher	Very Unlikely
2	1	Male	25-39	Traditional	Low	Lower	Neutral
3	1	Male	25-39	Traditional	High	Higher	Neutral
4	1	Male	25-39	Traditional	High	Lower	Very Likely
5	1	Male	25-39	Modern	Low	Higher	Somewhat Unlikely
6	1	Male	25-39	Modern	Low	Lower	Somewhat Unlikely
7	1	Male	25-39	Modern	High	Higher	Very Likely
8	1	Male	25-39	Modern	High	Lower	Very Likely
9	2	Female	16-24	Traditional	Low	Higher	Somewhat Unlikely
10	2	Female	16-24	Traditional	Low	Lower	Neutral
11	2	Female	16-24	Traditional	High	Higher	Very Likely
12	2	Female	16-24	Traditional	High	Lower	Very Likely

Data View Variable View

SPSS Processor is ready

Figure 8-1: Partial Listing of Conjoint Data File

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'.

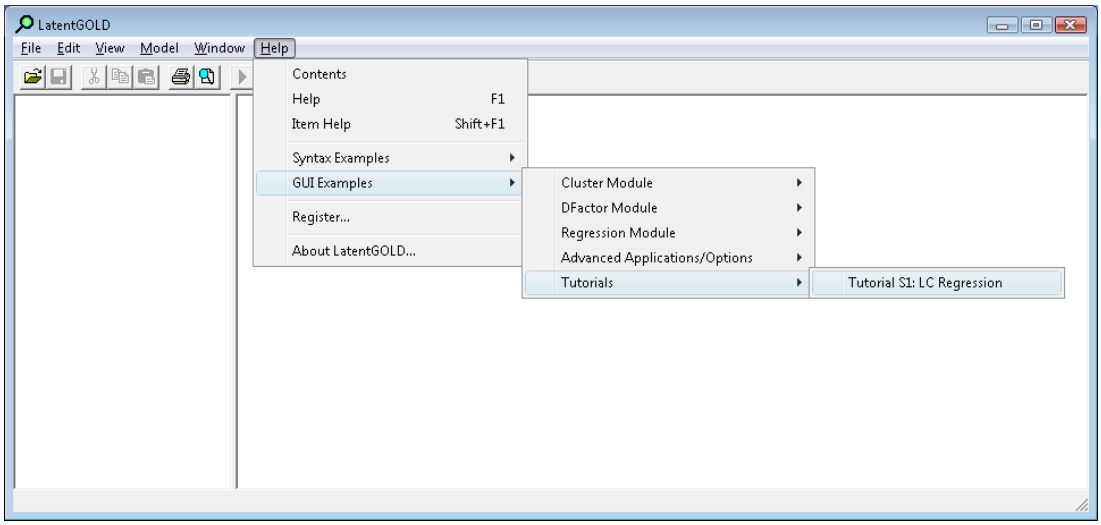


Figure 8-2: Examples Menu

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.

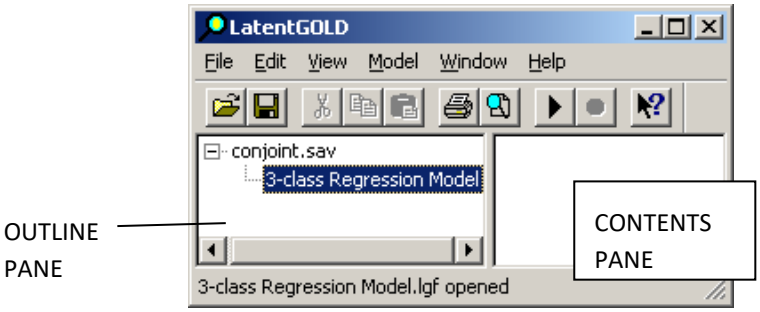


Figure 8-3: Outline and Contents Pane

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

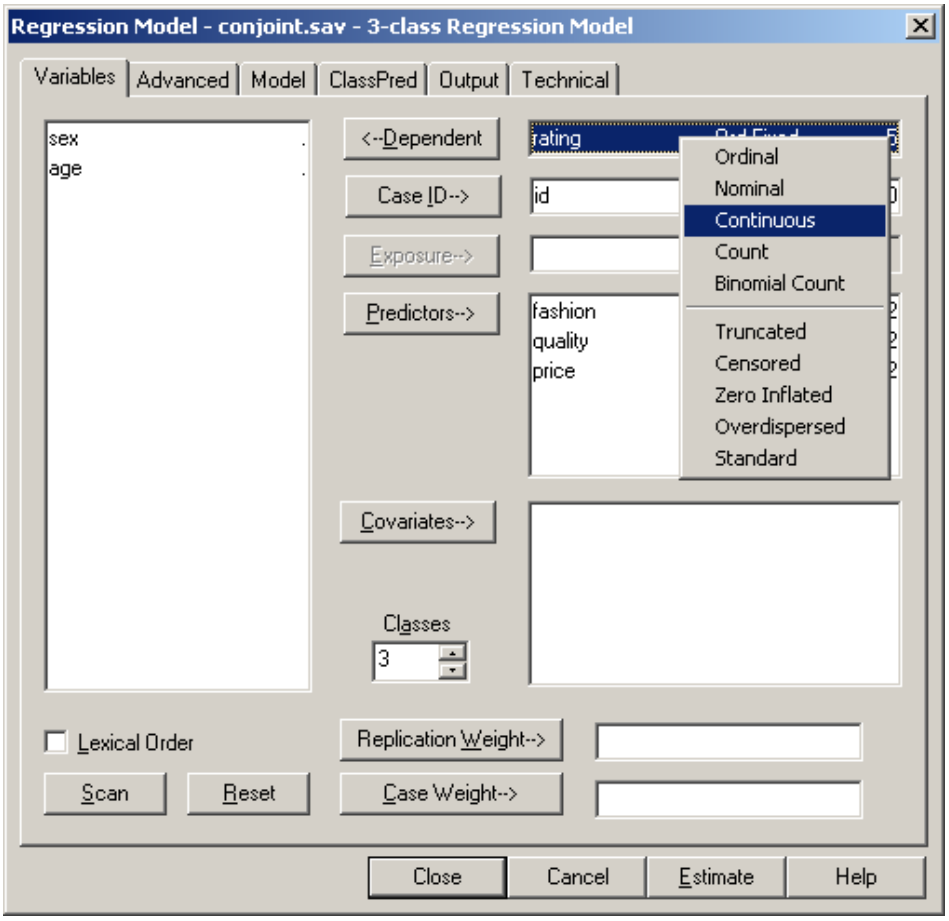


Figure 8-4: GUI Model Settings on the Analysis Dialog Box

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)

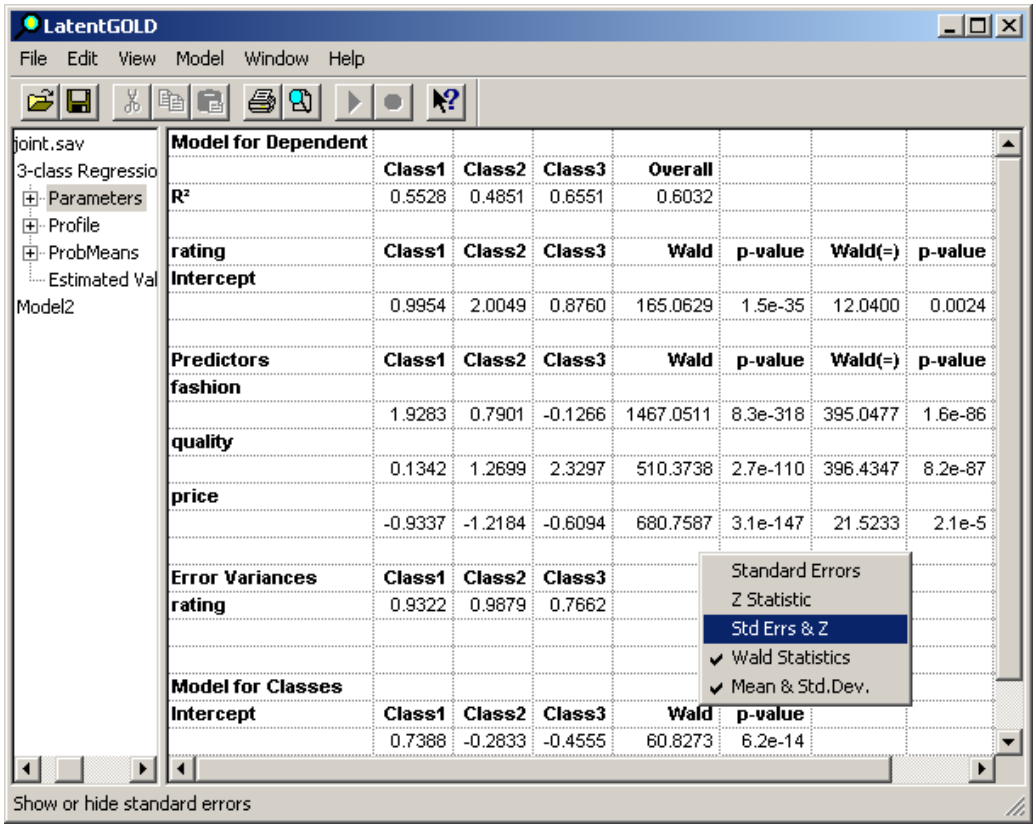
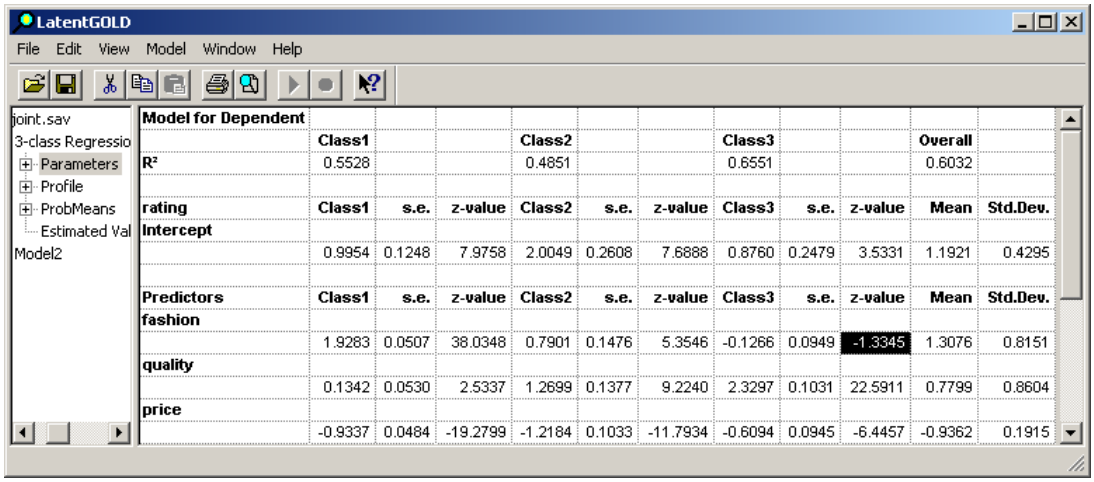


Figure 8-5: GUI Parameters Output

- 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).



Model for Dependent											
	Class1			Class2			Class3			Overall	
R^2	0.5528			0.4851			0.6551			0.6032	
rating	**Class1**	**s.e.**	**z-value**	**Class2**	**s.e.**	**z-value**	**Class3**	**s.e.**	**z-value**	**Mean**	**Std.Dev.**
Intercept											
	0.9954	0.1248	7.9758	2.0049	0.2608	7.6888	0.8760	0.2479	3.5331	1.1921	0.4295
Predictors	**Class1**	**s.e.**	**z-value**	**Class2**	**s.e.**	**z-value**	**Class3**	**s.e.**	**z-value**	**Mean**	**Std.Dev.**
fashion											
	1.9283	0.0507	38.0348	0.7901	0.1476	5.3546	-0.1266	0.0949	-1.3345	1.3076	0.8151
quality											
	0.1342	0.0530	2.5337	1.2699	0.1377	9.2240	2.3297	0.1031	22.5911	0.7799	0.8604
price											
	-0.9337	0.0484	-19.2799	-1.2184	0.1033	-11.7934	-0.6094	0.0945	-6.4457	-0.9362	0.1915

Figure 8-6: GUI Parameters Output Formatted to Include Z-values

- 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

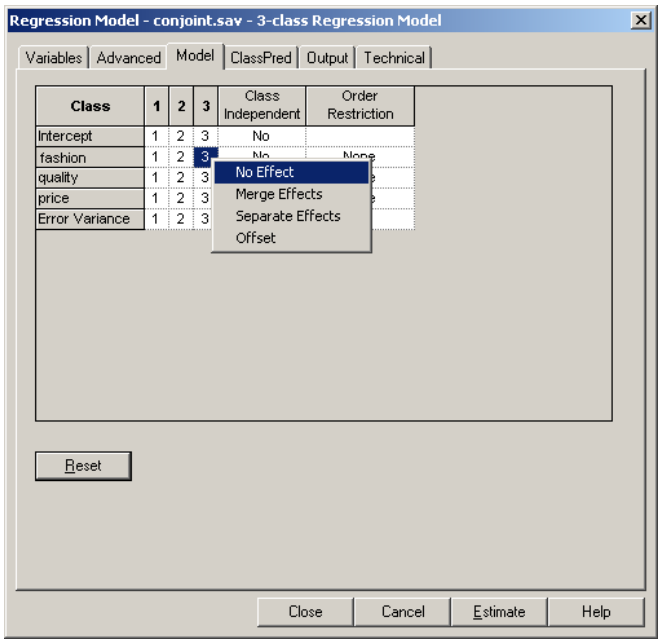


Figure 8-7: GUI Model Tab

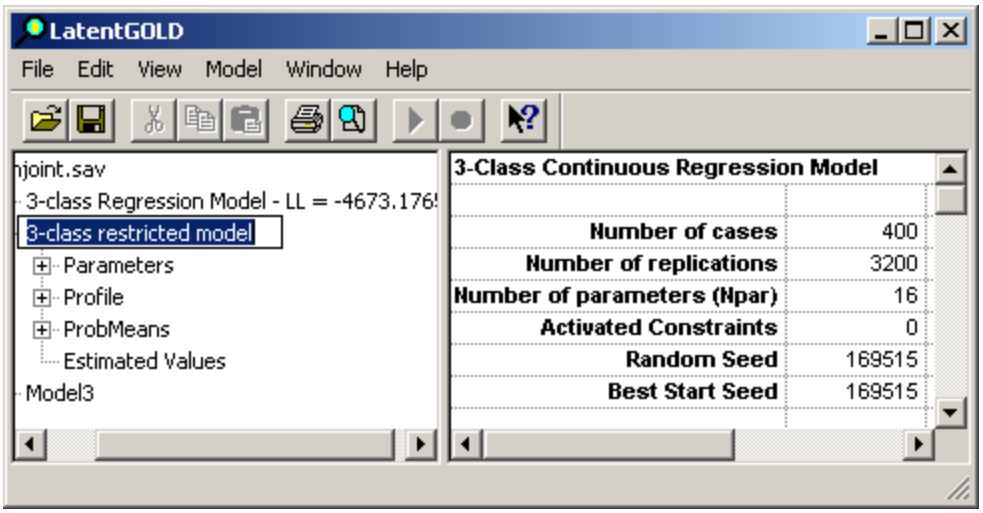


Figure 8-8: Renaming a Model

- 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.

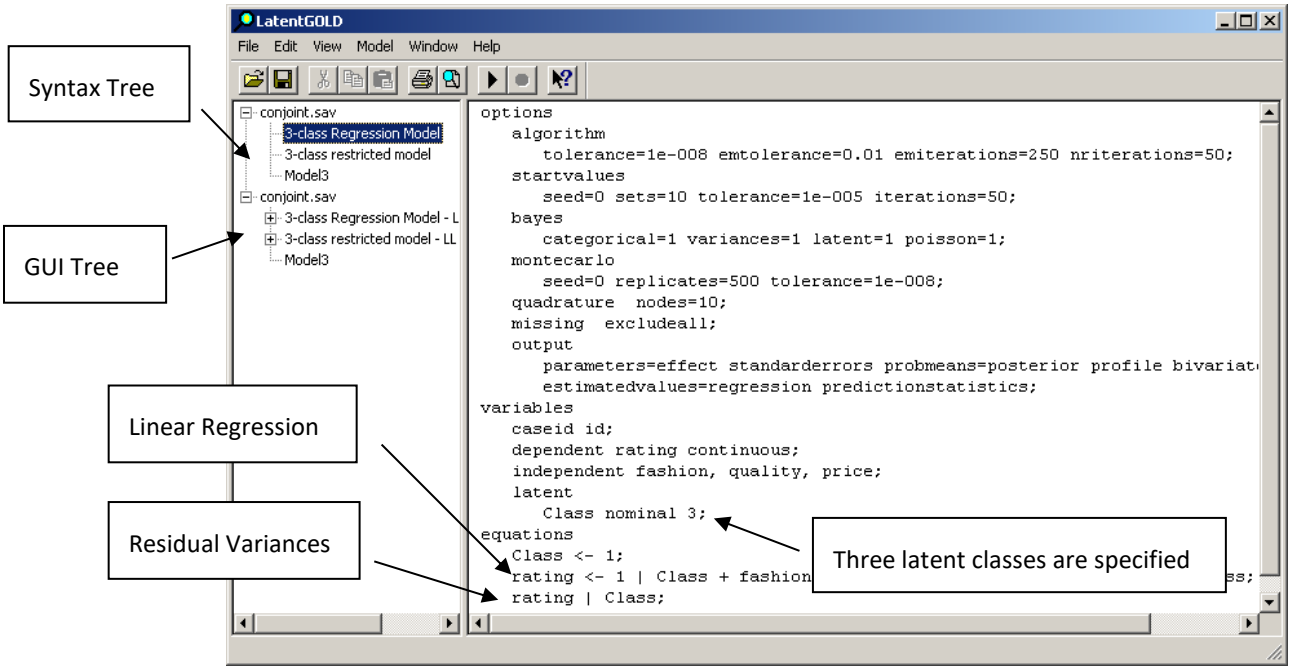


Figure 8-9: Syntax Specifications for Unrestricted 3-class Regression Model

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

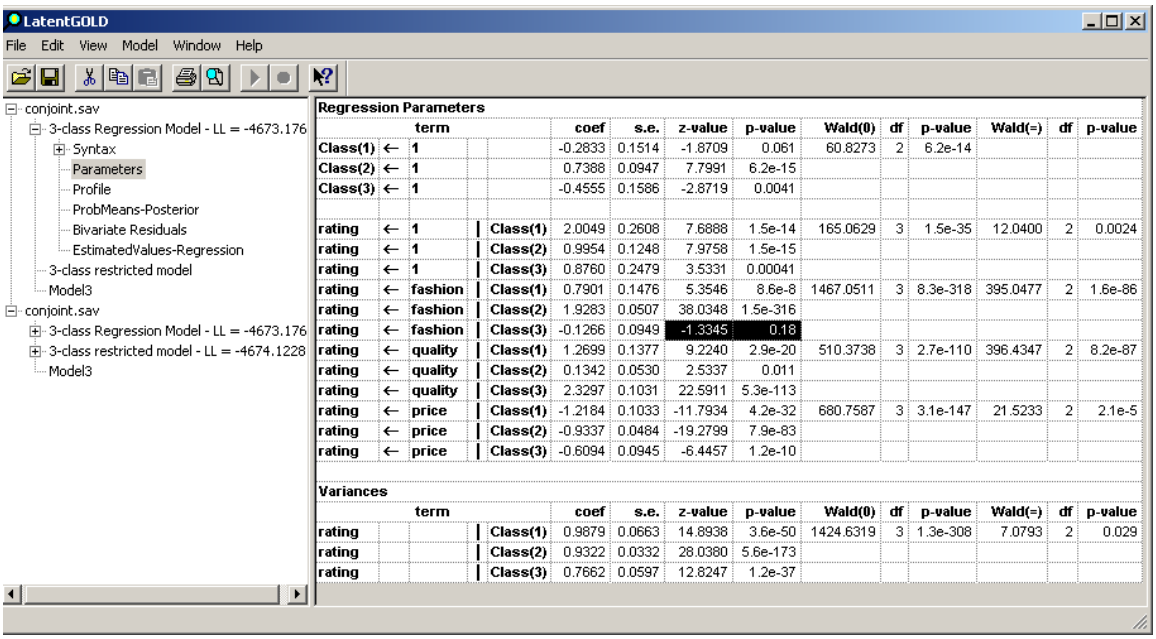
Upon completion of the estimation, the log-likelihood (LL) appears in the Outline Pane to the right of the model name (see Figure 8-10), and the Syntax and output listings appear as separate entries in the expanded Syntax tree. The model output listings in this expanded Syntax tree appear somewhat differently 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).



Regression Parameters

		term		coef	s.e.	z-value	p-value	Wald(0)	df	p-value	Wald(=)	df	p-value
Class(1)	←	1		-0.2833	0.1514	-1.8709	0.061	60.8273	2	6.2e-14			
Class(2)	←	1		0.7388	0.0947	7.7991	6.2e-15						
Class(3)	←	1		-0.4555	0.1586	-2.8719	0.0041						
rating	←	1	Class(1)	2.0049	0.2608	7.6888	1.5e-14	165.0629	3	1.5e-35	12.0400	2	0.0024
rating	←	1	Class(2)	0.9954	0.1248	7.9758	1.5e-15						
rating	←	1	Class(3)	0.8760	0.2479	3.5331	0.00041						
rating	←	fashion	Class(1)	0.7901	0.1476	5.3546	8.6e-8	1467.0511	3	8.3e-318	395.0477	2	1.6e-86
rating	←	fashion	Class(2)	1.9283	0.0507	38.0348	1.5e-316						
rating	←	fashion	Class(3)	-0.1266	0.0949	-1.3345	0.18						
rating	←	quality	Class(1)	1.2699	0.1377	9.2240	2.9e-20	510.3738	3	2.7e-110	396.4347	2	8.2e-87
rating	←	quality	Class(2)	0.1342	0.0530	2.5337	0.011						
rating	←	quality	Class(3)	2.3297	0.1031	22.5911	5.3e-113						
rating	←	price	Class(1)	-1.2184	0.1033	-11.7934	4.2e-32	680.7587	3	3.1e-147	21.5233	2	2.1e-5
rating	←	price	Class(2)	-0.9337	0.0484	-19.2799	7.9e-83						
rating	←	price	Class(3)	-0.6094	0.0945	-6.4457	1.2e-10						

Variances

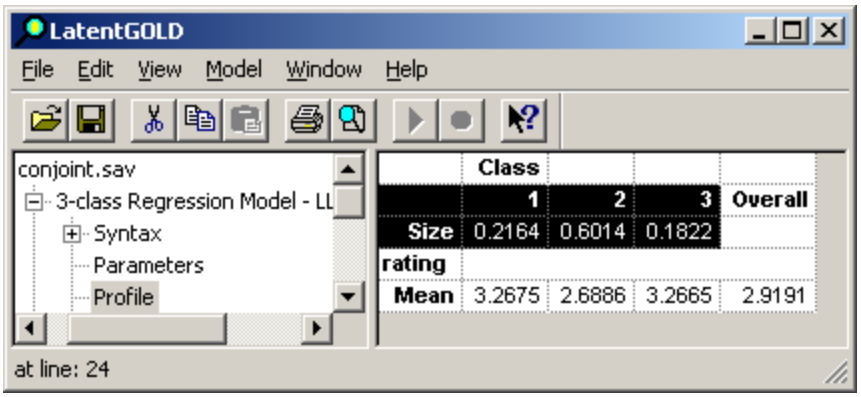
	term		coef	s.e.	z-value	p-value	Wald(0)	df	p-value	Wald(=)	df	p-value
rating		Class(1)	0.9879	0.0663	14.8938	3.6e-50	1424.6319	3	1.3e-308	7.0793	2	0.029
rating		Class(2)	0.9322	0.0332	28.0380	5.6e-173						
rating		Class(3)	0.7662	0.0597	12.8247	1.2e-37						

Figure 8-10: Syntax Parameters Model Output

While in models estimated in the GUI, the latent classes in an unrestricted model will automatically be ordered from large to small, in Syntax models this requires using the 'reorderclasses' output option. 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.



	Class			
	1	2	3	Overall
Size	0.2164	0.6014	0.1822	
rating				
Mean	3.2675	2.6886	3.2665	2.9191

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.

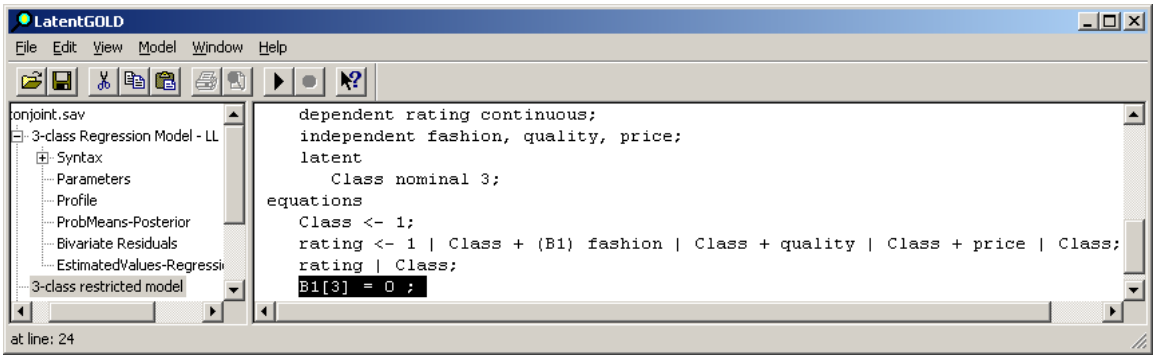


Figure 8-12: Syntax Equations for 3-class Restricted Model

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 on the toolbar
- Select the data file name to see how the models compare.

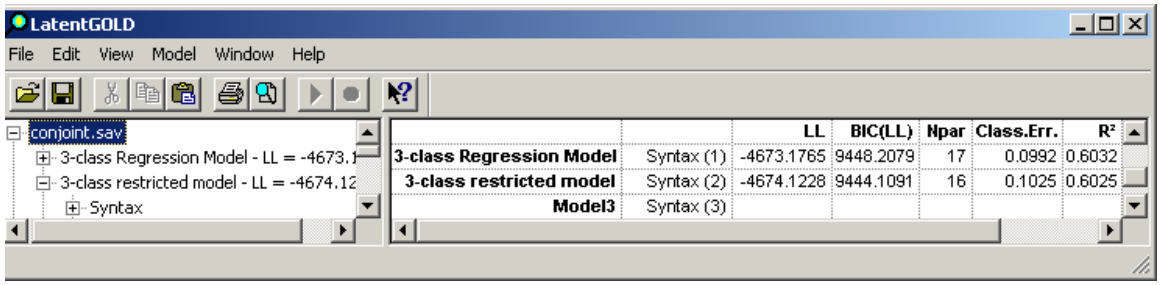
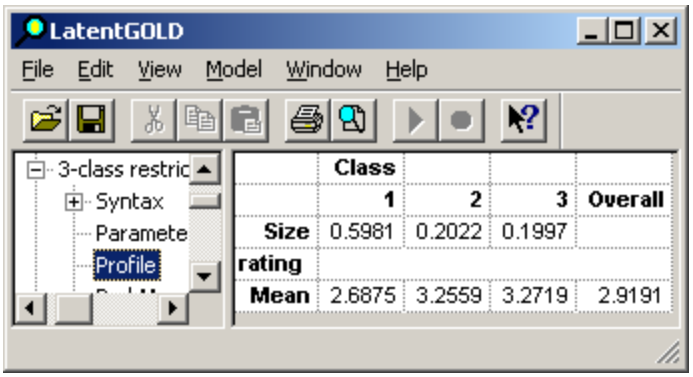


Figure 8-13: Model Summary Display

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:

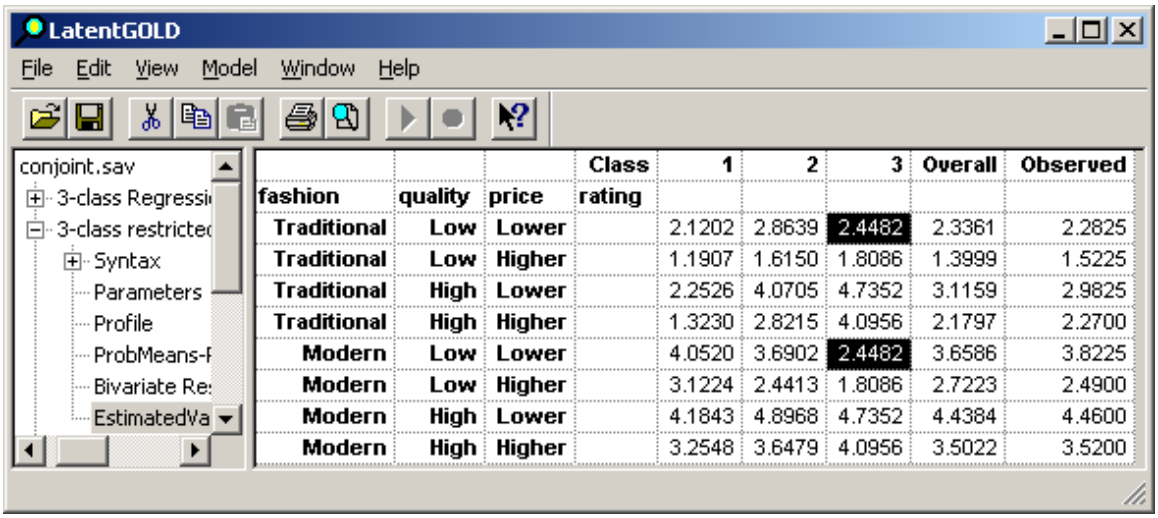


	Class			
	1	2	3	Overall
Size	0.5981	0.2022	0.1997	
rating				
Mean	2.6875	3.2559	3.2719	2.9191

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.



			Class	1	2	3	Overall	Observed
fashion	quality	price	rating					
Traditional	Low	Lower		2.1202	2.8639	2.4482	2.3361	2.2825
Traditional	Low	Higher		1.1907	1.6150	1.8086	1.3999	1.5225
Traditional	High	Lower		2.2526	4.0705	4.7352	3.1159	2.9825
Traditional	High	Higher		1.3230	2.8215	4.0956	2.1797	2.2700
Modern	Low	Lower		4.0520	3.6902	2.4482	3.6586	3.8225
Modern	Low	Higher		3.1224	2.4413	1.8086	2.7223	2.4900
Modern	High	Lower		4.1843	4.8968	4.7352	4.4384	4.4600
Modern	High	Higher		3.2548	3.6479	4.0956	3.5022	3.5200

Figure 8-15: EstimatedValues Regression Output

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.

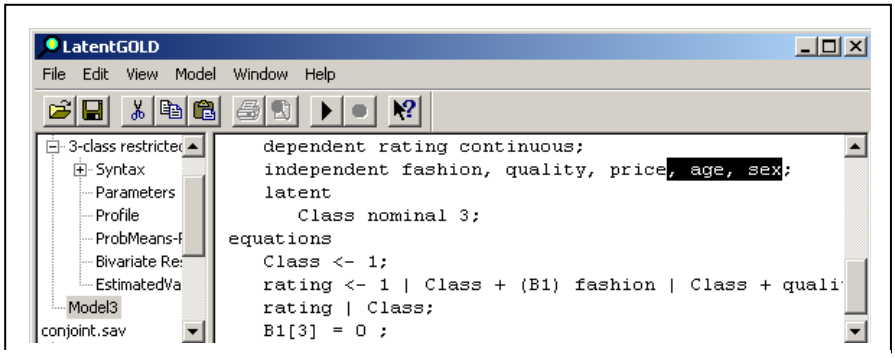


Figure 8-16B: Including Covariates in Syntax Editor

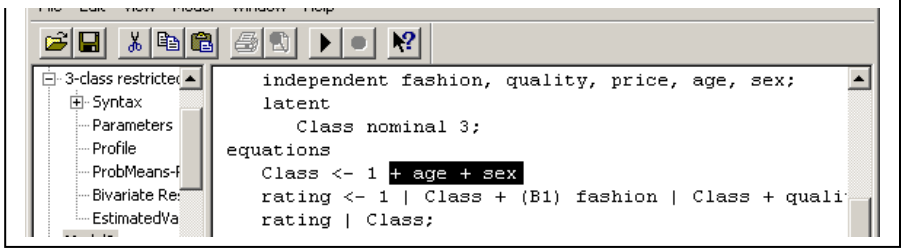
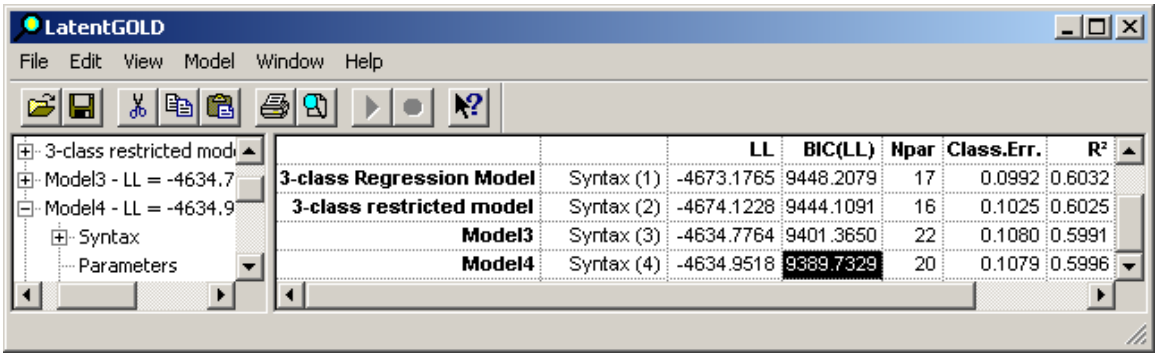


Figure 8-16A: List of Data File Variables

Figure 8-16C: Including Main and Interactive Effects of Covariates

- 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.

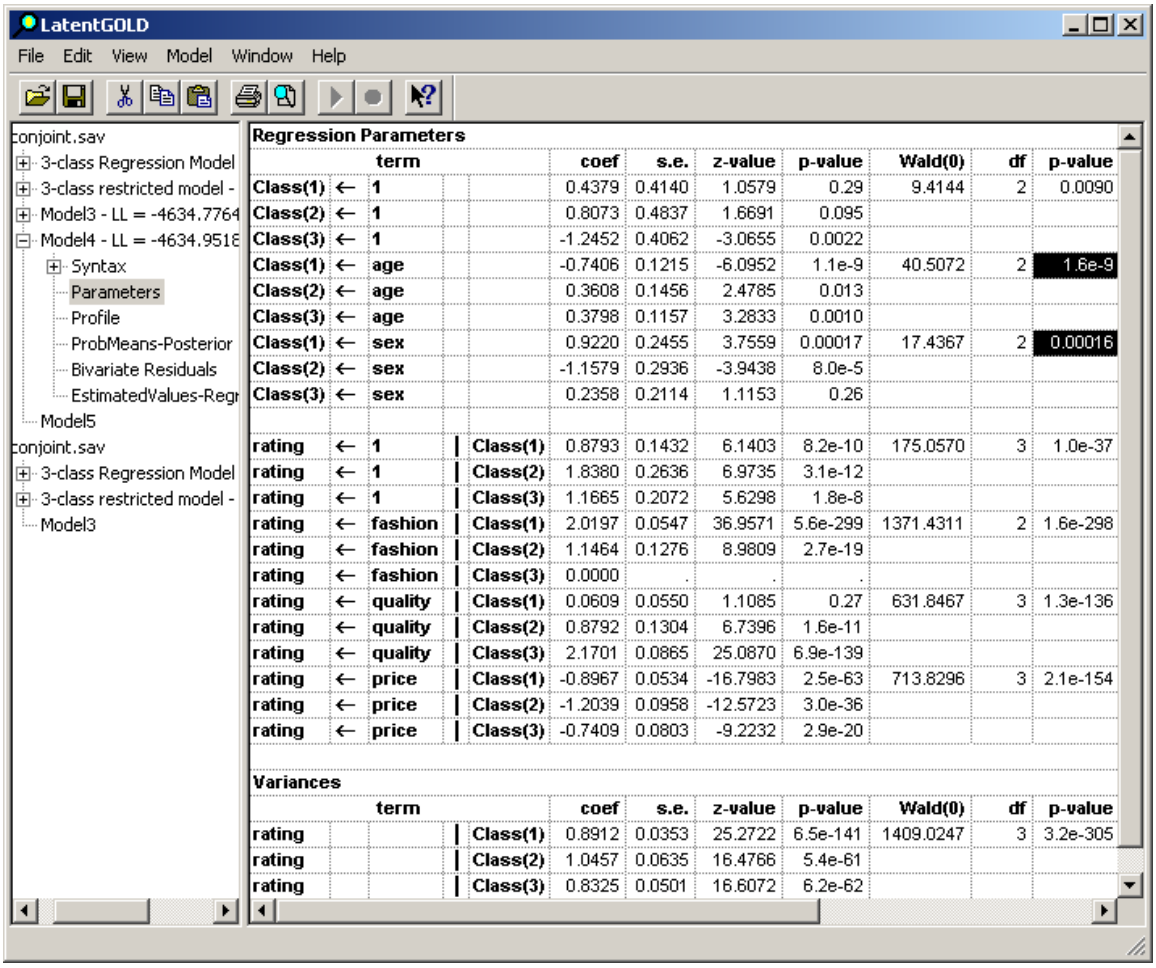


		LL	BIC(LL)	Npar	Class.Err.	R^2
3-class Regression Model	Syntax (1)	-4673.1765	9448.2079	17	0.0992	0.6032
3-class restricted model	Syntax (2)	-4674.1228	9444.1091	16	0.1025	0.6025
Model3	Syntax (3)	-4634.7764	9401.3650	22	0.1080	0.5991
Model4	Syntax (4)	-4634.9518	9389.7329	20	0.1079	0.5996

Figure 8-17: Model Summary Comparison of all 4 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.



Regression Parameters

		term		coef	s.e.	z-value	p-value	Wald(0)	df	p-value
Class(1)	←	1		0.4379	0.4140	1.0579	0.29	9.4144	2	0.0090
Class(2)	←	1		0.8073	0.4837	1.6691	0.095			
Class(3)	←	1		-1.2452	0.4062	-3.0655	0.0022			
Class(1)	←	age		-0.7406	0.1215	-6.0952	1.1e-9	40.5072	2	1.6e-9
Class(2)	←	age		0.3608	0.1456	2.4785	0.013			
Class(3)	←	age		0.3798	0.1157	3.2833	0.0010			
Class(1)	←	sex		0.9220	0.2455	3.7559	0.00017	17.4367	2	0.00016
Class(2)	←	sex		-1.1579	0.2936	-3.9438	8.0e-5			
Class(3)	←	sex		0.2358	0.2114	1.1153	0.26			
rating	←	1	Class(1)	0.8793	0.1432	6.1403	8.2e-10	175.0570	3	1.0e-37
rating	←	1	Class(2)	1.8380	0.2636	6.9735	3.1e-12			
rating	←	1	Class(3)	1.1665	0.2072	5.6298	1.8e-8			
rating	←	fashion	Class(1)	2.0197	0.0547	36.9571	5.6e-299	1371.4311	2	1.6e-298
rating	←	fashion	Class(2)	1.1464	0.1276	8.9809	2.7e-19			
rating	←	fashion	Class(3)	0.0000	.	.	.			
rating	←	quality	Class(1)	0.0609	0.0550	1.1085	0.27	631.8467	3	1.3e-136
rating	←	quality	Class(2)	0.8792	0.1304	6.7396	1.6e-11			
rating	←	quality	Class(3)	2.1701	0.0865	25.0870	6.9e-139			
rating	←	price	Class(1)	-0.8967	0.0534	-16.7983	2.5e-63	713.8296	3	2.1e-154
rating	←	price	Class(2)	-1.2039	0.0958	-12.5723	3.0e-36			
rating	←	price	Class(3)	-0.7409	0.0803	-9.2232	2.9e-20			

Variances

	term		coef	s.e.	z-value	p-value	Wald(0)	df	p-value
rating		Class(1)	0.8912	0.0353	25.2722	6.5e-141	1409.0247	3	3.2e-305
rating		Class(2)	1.0457	0.0635	16.4766	5.4e-61			
rating		Class(3)	0.8325	0.0501	16.6072	6.2e-62			

Figure 8-18: Parameters Output Showing Significance of Main Effects for Covariates (highlighted)

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.

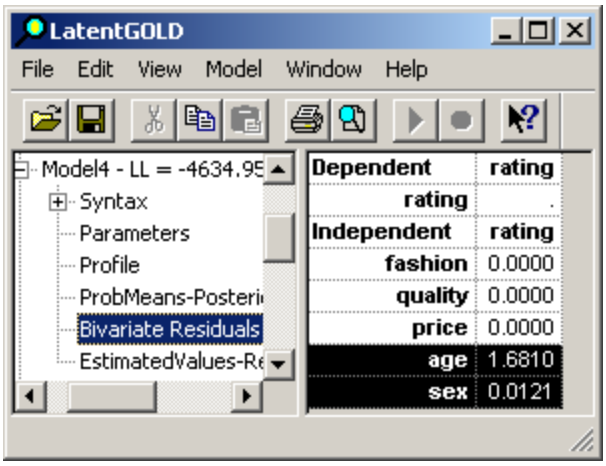


Figure 8-19: Bivariate Residuals – Regression Display for Model with Covariates

REFERENCES

Agresti, A. (2002). *Categorical data analysis*. Hoboken, NJ: Wiley.

Bakk, Z. and Kuha, J. (2018). Two-step estimation of models between latent classes and external variables. *Psychometrika*, 83, 871-892.

Bakk, Z., Oberski, D.L., and Vermunt, J.K. (2014). Relating latent class assignments to external variables: standard errors for corrected inference. *Political Analysis,* 22, 520-540*.*

Bakk, Z., Oberski, D.L., and Vermunt, J.K. (2016). Relating latent class membership to continuous distal outcomes: improving the LTB approach and a modified three-step implementation. *Structural Equation Modeling*, 23, 278-289

Bakk, Zs., Tekle, F.B., and Vermunt, J.K. (2013). Estimating the association between latent class membership and external variables using bias adjusted three-step approaches. *Sociological Methodology*, 43, 272-311.

Bakk, Zs, and Vermunt, J.K. (2016). Robustness of stepwise approaches to latent class modeling with continuous distal outcomes. *Structural Equation Modeling*, 23, 20-31

Böckenholt, U. (2005). A latent Markov model for the analysis of longitudinal data collected in continuous time: States, durations, and transitions, *Psychological Methods*, 10, 65–83.

Bolck, A., Croon, M.A., and Hagenaars, J.A. (2004). Estimating latent structure models with categorical variables: One-step versus three-Step estimators. *Political Analysis*, 12, 3–27.

Brančík, L. (2006). Techniques of matrix exponential function derivative for electrical engineering simulations, Proceedings of 2006 IEEE International Conference on Industrial Technology, 2608- 2613.

Chorus, C.G. (2010). A new model of Random Regret Minimization. *European Journal of Transport and Infrastructure Research*, 10, 181-196.

Chorus, C.G. (2012). Random Regret-based discrete choice modeling: A tutorial. Heidelberg: Springer.

Celeux, G., and Govaert, G. (1992). *A classification EM algorithm for clustering and two stochastic versions. Computational Statistics & Data Analysis*, 14, 315--332.

Clogg, C. C., and Eliason, S. R. (1987). Some common problems in log-linear analysis. *Sociological Methods Research*, 15, 4-44.

Bray, B.C., Dziak, J.J., Patrick, M.E., and Lanza, S.T. (2019). Inverse propensity score weighting with a latent class exposure: Estimating the causal effect of reported reasons for alcohol use on problem alcohol use 16 years later. *Prevention Science*, 20, 394-406.

Clouth, F.J., Pauws, S., Mols, F., and Vermunt. J.K. (2022). Three-step latent class analysis with inverse propensity weighting in the presence of differential item functioning. *Advances in Classification and Data Analysis*, 16, 351-371.

Cox, D.R., and Miller, H.D. (1965). *The theory of stochastic processes.* London: Methuen.

Crayen, C., Eid, M., Lischetzke, T.; Courvoisier, D.S., and Vermunt, J.K. (2012). Exploring dynamics in mood regulation - mixture latent Markov modeling of ambulatory assessment data. *Psychosomatic Medicine*, 74, 366-376.

de Boor, C. (1978). A Practical Guide to Splines. Springer-Verlag.

De Roover, K. (2021). Finding clusters of groups with measurement invariance: Unraveling intercept non-invariance with mixture multigroup factor analysis. *Structural Equation Modeling*, 28, 663-683.

De Roover, K., and Vermunt, J.K. (2019). On the exploratory road to unravelling factor loading non-invariance: A new multigroup rotation approach, *Structural Equation Modeling*, 26, 905-923.

De Roover, K., Vermunt, J.K., and Ceulemans, E. (2022). Mixture multigroup factor analysis for unraveling factor loading non-invariance across many groups, *Psychological Methods*, 27, 281-306 Di Mari, R., and Bakk, Z. (2018). Mostly harmless direct effects: A comparison of different latent Markov modeling approaches. *Structural Equation Modeling*, 25(3), 467-483.

Francis, B., Elliott, A., and Weldon, M. (2016). Smoothing group-based trajectory models through b-splines. *Journal of Developmental and Life-Course Criminology*, 2 (1), 113–133.

Gudicha, D.W., Tekle, F.B., and Vermunt, J.K. (2016). Power and sample size determination for latent class models. *Journal of Classification*, 33, 30-51

Gudicha, D.W., Schmittmann, V.D., and Vermunt, J.K. (2016). Power computation for likelihood-ratio tests for the transition parameters in latent Markov models. *Structural Equation Modeling*, 23, 234-245

Jackson C.H., and Sharples. L.D. (2002). Hidden Markov models for the onset and progression of bronchiolitis obliterans syndrome in lung transplant recipients. *Statistics in Medicine*, 21, 113–128. 2002.

Jennrich, R. I. (1973). Standard errors for obliquely rotated factor loadings. *Psychometrika*, 38, 593-604.

Jennrich, R. I. (2002). A simple general method for oblique rotation. *Psychometrika*, 67, 7-19.

Kalbfleisch, J.D., and Lawless, J.F. (1985). The analysis of panel data under a Markov assumption. *Journal of the American Statistical Association*, 80, 863–871.

van Kollenburg, G.H., Mulder, J., and Vermunt, J.K. (2018). The lazy bootstrap. A fast resampling method for evaluating latent class model fit. Working paper.

Lanza, T. S., Tan, X., and Bray, C. B. (2013). Latent class analysis with distal outcomes: A flexible model-based approach. *Structural Equation Modeling*, 20, 1-26.

Lattery, K. (2010). Anchoring maximum difference scaling against a threshold – Dual response and direct binary responses. Sawtooth Software Research Paper Series.

Lê, T.T., Clouth, F.J., and Vermunt, J.K. (2025). Causal latent class analysis with distal outcomes: A modified three-step method using inverse propensity weighting. *Multivariate Behavioral Research*.

Lo, Y., Mendell, N.R., and Rubin, D. B. (2001). Testing the number of components in a normal mixture. *Biometrika*, 88, 767–778.

Long, J. S. (1997). *Regression models for categorical and limited dependent variables*, vol. 7 of Advanced Quantitative Techniques in the Social Sciences. Thousand Oaks, CA: Sage.

Long, J.S., Freese, J. (2014). *Regression models for categorical dependent variables using Stata*. 3rd ed. College Station, TX: Stata Press.

Magidson, J., Thomas, D., and Vermunt, J.K. (2009). A new model for the fusion of MaxDiff scaling and ratings data. 2009 Sawtooth Software Conference Proceedings, 83-103.

Magidson, J., and Vermunt, J.K. (2007). Removing the scale factor confound in multinomial logit choice models to obtain better estimates of preference. October 2007 Sawtooth Software Conference Proceedings.

Merkle, E.C., You, D., and Preacher, K J. (2016). Testing non-nested structural equation models. *Psychological Methods*, 21, 151–163.

Moler, C., and Van Loan, C. (2013). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Revie*w, 45, 3–49.

Nagelkerke, E., Oberski, D.L., and Vermunt, J.K. (2016). Goodness-of-fit of multilevel latent class models for categorical data. *Sociological Methodology*, 46, 252-28.

Nagelkerke, E. (2018). Local fit in multilevel latent class and hidden Markov models (PhD Dissertation). Vianen: Proefschriftmaken.

Oberski, D.L. (2014), Evaluating Sensitivity of Parameters of Interest to Measurement Invariance in Latent Variable Models, *Political Analysis*, 22, 45-60.

Oberski, D.L, van Kollenburg, G.H., and Vermunt, J.K. (2013). A Monte Carlo evaluation of three methods to detect local dependence in binary data latent class models. *Advances in Classification and Data Analysis*, 7, 267-279.

Oberski, D.L., and Vermunt, J.K. (2013). A model-based approach to goodness-of-fit evaluation in item response theory. *Measurement*, 3, 117-122.

Oberski, D.L., and Vermunt, J.K. (2016). The expected parameter change (EPC) for local dependence assessment in binary data latent class models. Unpublished working paper.

Oberski, D.L., Vermunt, J.K., and Moors, G.B.D. (2015). Evaluating measurement invariance in categorical data latent variable models with the EPC-interest, *Political Analysis*, 23, 550-563.

Orme, B.K. (2009). Anchored scaling in MaxDiff using dual response. Sawtooth Software Research Paper Series.

Schneider, L., Chalmers, R.P., Debelak R., and Merkle, E.C. (2020). Model selection of nested and non-nested item response models using Vuong tests, *Multivariate Behavioral Research*, 55, 664-684,

Sharples, L.D., Jackson, C.H., Parameshwar, J., Wallwork, J., Large, S.R. (2003). Diagnostic accuracy of coronary angiography and risk factors for post-heart-transplant cardiac allograft vasculopathy. *Transplantation*, 76, 679–682.

Tay, L., Vermunt, J.K., and Wang, C. (2013). Assessing the item response theory with covariate (IRT-C) procedure for ascertaining differential item functioning. *International Journal of Testing*, 13, 201-222.

van den Bergh, M., Schmittmann, V.D., and Vermunt, J.K. (2017). Building latent class trees, with an application to a study of social capital. *Methodology*, 13(Supplement), 13–22.

van den Bergh, M., van Kollenburg, G.H., and Vermunt, J.K. (2018). Deciding on the starting number of classes of a latent class tree, *Sociological Methodology*, 48, 303–336..

Van der Palm, D.W., van der Ark, L.A., and Vermunt, J.K. (2016a). A comparison of incomplete-data methods for categorical data. *Statistical Methods in Medical Research*, 25, 754-774

Van der Palm, D.W., Van der Ark, L.A., and Vermunt, J.K. (2016b). Divisive latent class modeling as a density estimation method for categorical data. *Journal of Classification*, 33, 52-72

Vermunt, J.K. (1997). *Log-linear models for event histories*. Advanced Quantitative Techniques in the Social Sciences Series. Thousand Oakes: Sage Publications.

Vermunt, J.K. (2010). Latent class modeling with covariates: Two improved three-step approaches. *Political Analysis*, 18, 450-469.

Vermunt, J.K.. (2011). K-means may perform as well as mixture model clustering but may also be much worse: Comment on Steinley and Brusco (2011). *Psychological Methods*, 16, 82-88.

Vermunt, J.K. (2013). Categorical response data. In: M.A. Scott, J.S. Simonoff, and B.D. Marx (eds.), The SAGE *Handbook of Multilevel Modeling*, 287-298. Thousand Oaks, CA: Sage.

Vermunt, J.K. (2024). The Vuong-Lo-Mendell-Rubin test for latent class and latent profile analysis: A note on the different implementations in Mplus and LatentGOLD, *Methodology*, 20, 72-83.

Vermunt, J.K. (2025). Stepwise latent variable modeling: An overview of approaches. *Statistical Modelling*.

Vermunt, J.K., and Magidson, J. (2005). Factor Analysis with categorical indicators: A comparison between traditional and latent class approaches. In A. Van der Ark, M.A. Croon, and K. Sijtsma (eds.), *New Developments in Categorical Data Analysis for the Social and Behavioral Sciences*, 41-62. Mahwah: Erlbaum.

Vermunt, J.K., and Magidson, J. (2021). How to perform three-step latent class analysis in the presence of measurement non-invariance or differential item functioning, *Structural Equation Modeling*, 28, 356-364.

Vermunt, J.K., and Magidson, J. (2025). Linear logistic scoring equations for latent class and latent profile models: A simple method for classifying new cases. *Structural Equation Modeling*.

Vermunt, J.K., Van Ginkel, J.R., Van der Ark, and L.A., Sijtsma K. (2008). Multiple imputation of categorical data using latent class analysis. *Sociological Methodology*, 33, 369-297.

Vidotto, D., Kaptein, M.C., and Vermunt, J.K. (2015). Multiple imputation of missing categorical data using latent class models: State of art. *Psychological Test and Assessment Modeling*, 57, 542-576.

Vidotto, D., Vermunt, J.K., and Van Deun, K., (2018). Bayesian multilevel latent class models for the multiple imputation of nested categorical data. *Journal of Educational and Behavioral Statistics*, 43, 511-539.

Vidotto, D., Vermunt, J.K., and Van Deun, K., (2020). Multiple Imputation of longitudinal categorical data through Bayesian mixture latent Markov models, *Journal of Applied Statistics*, 47, 1720-1738.

Vogelsmeier, L.V.D.E., Vermunt, J.K., Böing-Messing, F., and De Roover, K. (2019). Continuous-time latent Markov factor analysis for exploring measurement model changes across time, *Methodology*,15(Supplement), 29-42.

Vuong, Q. H. (1989). Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica*, 57(2), 307–333.

_

Constants · 60

A

adjacent · 16, 21, 39, 53
aft · 38, 39
aggregate · 24, 31
algorithm · 23, 26, 28, 37
allchi2 · 24, 29, 99
alpha · 17, 24, 29
alternatives · 59
alternative-specific independent variables · 59
anchoritem · 60
anneal · 24, 28
append · 19, 25, 30, 34, 98, 107, 108
ar1 · 53, 54
association · 23, 46, 47, 66
associationstatistics · 25, 34
attributes · 59, 60

B

Bakk-Kuha · 44, 79
batch mode · 8, 13, 14
bayes · 24, 28, 37
bch · 24, 26, 30, 35, 36, 78, 109
bestworst · 59
beta · 16, 38, 39, 40, 77
BHHH · 21, 23, 26, 27
binomial · 21, 38, 39, 40
BLRT · 29, 100, 101
bootstrap · 17, 18, 19, 29, 34, 35, 37, 99, 105, 108
bray · 24, 31
bspline · 41, 60, 81
bvr · 25, 37
bvrlongitudinal · 25
bvrtwolevel · 25

C

case · 32, 33, 38, 39, 42
caseid · 37, 38, 72, 98
caseweight · 38, 95
categorical · 24, 37
cem · 23, 27
censored · 38, 39
centered · 44
certain · 44
choice · 58, 59, 86
choicesetid · 59
choicesets · 59
Cholesky · 32, 56, 92, 93
classification · 25, 26, 32, 35, 36, 37, 94
classification EM algorithm · 21
classification tables · 32
Classification-Forward · 33
coding · 20, 35, 38, 40, 41, 42, 43, 52, 75
complex sampling · 7, 19, 35, 44, 90, 104, 105
continuous · 38, 39, 40, 41, 42, 43
continuous latent variables · 15, 16, 21, 30, 32, 33, 36, 43, 46, 47, 56, 67, 68
continuous-time latent Markov · 44, 73, 76, 82, 83
cooksD · 26
cor · 53, 54
covariance · 23, 46, 47
Cox model · 38, 39, 40
ct · 28
cumlogit · 38, 39, 40, 77, 84
cumulative logit model · 76
CV · 24, 29

D

degree · 41, 60
dependent · 32, 38, 39, 41, 46, 59
dia · 53, 54
dif · 42, 43, 52, 53
Dirichlet · 21, 37, 38, 39, 40, 54, 77
discrete-time · 43
distribution · 42, 43
divisive · 26, 37
dpp · 37
dynamic · 16, 32, 33, 42, 52, 72

E

EFA · 85
effect · 20, 25, 35
eigen · 23, 27
em · 26, 37
emfa · 23, 27
emiterations · 23, 24, 30, 37, 91
emtolerance · 23, 24, 30, 37
equations · 15, 22, 23, 46, 55
err · 52, 53, 73, 74, 75
estimatedvalues · 25, 34, 37
event · 38, 39
excludeall · 24
exploratory factor analysis · 16, 85
expm · 23
exponential · 39, 40

exposure · 21, 38, 40, 41
expressions · 23, 26, 34, 57, 58, 102, 103, 105

F

factor analysis · 16
factorscores · 26, 36
fast · 25, 35
first · 20, 25, 35, 37, 38, 41, 42, 43
Fisher scoring · 27
forward · 25, 26, 36
frequencies · 25, 35
ful · 53, 54
fused models · 87

G

gamma · 16, 38, 39, 40, 77
geomin · 25, 31
gibbs · 26, 37
gompertz · 38, 39, 40
group · 16, 32, 38, 42, 43, 64
groupid · 37, 38, 75, 80
group-level weight · 20
groupweight · 38, 95

H

halfnormal · 42, 43
hblike · 25, 26, 35, 36, 107
hessian · 23, 26, 27
holdout · 15, 18, 45, 107

I

id · 26, 45
identification · 19, 25, 34, 35, 37, 105
ignoreclassification · 38, 41
ignoreknownclass · 25, 35
importance · 59
imputation · 15, 18, 26, 29, 36, 95
imputation_# · 18, 36, 95, 96
imputationid · 18, 36, 45, 95, 96
inactive · 41, 42
includeall · 24, 28, 37, 72
includedependent · 24
independent · 21, 32, 41, 42
individual best-worst scores · 59
individualcoefficients · 26, 36
infile · 22, 94
infmat · 23, 26
int · 53, 54, 83
intercept · 53, 54, 83
internal parameters · 35, 55, 90, 91, 92
ipw · 44
IRT model · 67, 68
item response models · 16
iterationdetails · 24, 30, 37
iterations · 24, 37, 91

J

jackknife · 19, 25, 29, 34, 35, 104, 105
joint · 60

K

keep · 26, 37
knots · 41, 60
knownclass · 42, 43, 76, 83, 84

L

L2 · 24, 29, 99
last · 25, 35, 41, 43, 75
latent · 42
latent Markov models · 16, 72
LC tree · 21, 31, 43
Levenberg-Marquardt · 21, 27
LLdiff · 24, 25, 26, 29, 34, 99, 100, 103
LLdiffpower · 25
LLdiffPower · 34, 103, 104
LMR · 34, 100
loadings · 25
local dependencies · 64
logdensity · 26, 36, 44
log-linear scale · 17, 46, 47, 84, 87
log-log · 16, 39, 40, 76
loglog1 · 38, 39, 40, 76
loglog2 · 38, 39, 40, 76
loglogistic · 38, 39, 40
lognormal · 38, 39, 40
longitudinal · 25
LTB · 21, 25, 33

M

MAR assumption · 20
MARchi2 · 25, 34
marginal effects · 19, 25, 33
markov · 42
maxlevels · 25, 31
maxnodes · 25, 31
maxthreads · 23, 26, 37
MCstudy · 17, 24, 29, 30, 96, 97, 98, 99, 108
minBVR · 25, 31

minentropyR2 · 25, 31
minN · 25, 31
mis · 53, 54
missing · 21, 24, 28, 34, 37, 54, 83
missing values · 18, 20, 28, 54, 72, 90, 95
missingvaluedummy · 20, 21
miterations · 23, 27
mixture EFA · 86
mixture EFA model · 86
ml · 24
modal · 24, 30, 31
monotonicity restrictions · 46, 50, 55, 82
Monte Carlo simulation · 7, 15, 17, 19, 29, 90
montecarlo · 24, 29, 61, 96, 97, 99
mpoisson · 38, 39
MstepNR · 23, 26
multinomial · 38, 39, 40, 41
multiple imputation · 7, 18, 45, 90, 95
multiple-group · 15, 27, 64
multiple-group EFA · 85

N

nodes · 21, 24, 42, 43
noignoreclassification · 24, 30
nom · 52, 53
nominal · 20, 38, 39, 41, 42, 60
NoMstepNR · 23, 27
none · 24, 25, 30, 31
norescale · 24, 30
nowrite · 24, 30, 108
npbootstrap · 25, 35
nriterations · 23, 24, 27, 30, 37, 91
number of EM iterations · 27
numeric · 41, 60

O

oblimin · 25, 31, 85
obstype · 44
off · 53, 54
options · 9, 22, 23, 30, 35, 36, 37, 41, 44, 45, 107, 108
ord · 52, 53
ordinal · 38, 39, 40, 41, 42, 43
ordinal dependent variables · 16, 76
outfile · 26, 36, 37, 55, 61, 94, 95, 96, 98
Output · 8, 9, 10, 11, 13, 18, 32, 33, 90
overdispersed · 38, 39, 40, 41, 46, 47

P

pade · 23, 27
parameter restrictions · 48
ParameterCorrelations · 25, 33
ParameterCovariances · 25, 33
parameters · 25, 34, 35, 37, 55, 63, 82, 93
partition · 24, 28
pbootstrap · 25, 35, 99
pca · 24, 28
percategory · 24, 28
perclass · 24, 28
piecewise exponential · 39
poisson · 21, 24, 37, 38, 39, 40, 41
populationsize · 44
posterior · 24, 25, 26, 28, 35, 36, 42, 43, 78, 79, 80, 107
power · 17, 21, 24, 29, 30, 34, 55, 90, 96, 97, 98, 99, 103, 104, 108
power computation · 90, 103, 108
prediction · 26, 35, 36, 37, 94
prediction statistics · 18, 32, 35, 36, 45, 107
predictionstatistics · 25, 33
pri · 53, 54
price · 60
probit · 16, 38, 39, 40, 76
probmeans · 25, 34, 37
Probmeans-Model · 33
procrustes · 25, 31, 85
profile · 25, 34, 37
Profile-longitudinal · 20, 32
Profile-Posterior · 32
propensity · 31, 41, 42, 44
proportional · 24, 30, 78
psuid · 19, 44

Q

quadrature · 24, 30, 42

R

rank · 34, 38, 40, 41, 42
ranking · 59
rating · 59
regression · 23, 25, 38, 39, 46
regression equations · 46
reorderclasses · 25, 33
replicates · 17, 18, 24, 29, 35, 98, 105
replication scale · 59
replicationweight · 38, 95
repscale · 59
resampling · 29, 34
rescale · 44
response file · 59
robust · 25, 35
rotation · 16, 25, 31, 32
rrm · 60, 61
rsweights · 44
rum · 61

S

samplesizeBIC · 25
sampling weight · 19
samplingweight · 44
scale classes · 87
scaled · 44
score · 38, 40, 41, 42, 43
score test · 19, 34, 35, 90, 104, 108
scoretest · 26, 34
scoring · 7, 11, 15, 16, 20, 32, 33, 55, 78, 90, 94
scoring equation · 78, 109
ScoringEquations · 25, 33
seed · 24, 26, 27, 29, 36, 37
select · 45
seqlogit1 · 38, 39, 40, 76
seqlogit2 · 38, 39, 40, 76
sets · 24, 27, 29, 34, 37
setsprobmeans · 59
setsprofile · 59
SimulateParameters · 25, 34, 105
simulation · 17, 26, 29, 34, 36, 55, 90, 91, 96, 98, 99, 103
simulationid · 17, 36, 45, 96, 98
simultaneous · 24
special parameter coding · 52
split · 21, 31, 42, 43, 44
sqr · 53
standard · 25, 36, 98, 99
standard errors · 17, 18, 19, 21, 33, 34, 35, 45, 56, 58, 96, 99, 103, 104, 105
standarderrors · 25, 37, 99
starting values · 15, 17, 19, 23, 27, 28, 30, 46, 47, 51, 52, 54, 55, 56, 81, 83, 91, 96, 99, 109
startvalues · 24, 37
step3 · 24, 30, 78, 79, 80, 109
stratumID · 44
survey options · 44, 104

T

t0 · 38, 39
target · 25, 31, 32
timeid · 37, 38
timeinterval · 44, 73, 82, 83
timelabel · 44, 45
title · 22, 23
tolerance · 23, 24, 30, 37
tra · 52, 53, 73, 74, 75
transition · 32, 52, 82
transition probabilities · 27, 32, 43, 52, 53, 74, 81, 82, 98
tree · 24, 31, 42, 43, 44, 116
truncated · 38, 39, 40, 41

U

unequal · 60
uniform · 24, 28
upper · 38, 39
user-defined expressions · 19, 57, 102

V

validation · 7, 15, 18, 29, 34, 45, 107
validation log-likelihood · 18, 29, 34, 45
validationid · 18, 45, 107
validationLL · 25, 45, 107
variance · 23, 44, 46, 47
variances · 24, 37
varimax · 25, 31, 85
varweight · 38, 41
VLMR · 34, 100
von Mises · 16, 39, 40, 77
vonmises · 38, 39, 40
Vuong · 34, 99

W

Wald tests · 19, 20, 21, 32, 34, 57, 90, 101, 102, 103
waldpower · 25
Waldtest · 25, 34, 101
wei · 53, 54, 76, 80, 81
weibull · 38, 39, 40
weight · 54
write · 19, 25, 30, 34, 98, 107, 108
writeBCH · 19, 26
writeclassificationerrors · 26, 35, 56
writeclassificationerrorsproportional · 26, 35
writeexemplarydata · 19, 26, 35, 104
writefactorerrors · 26, 35, 108, 109
writeGensyntax · 26, 33, 95, 109
writegradients · 26
writehessian · 19, 26, 35
writeinternalparameters · 26, 35
writeparameters · 26, 35
WriteRSGHB · 26, 33
writeRsyntax · 26, 33, 95, 109
writeSPSSsyntax · 26, 33, 95, 109

Z

zeroinflated · 38, 39, 40