

BLIMP 3



User's Guide

BRIAN T. KELLER & CRAIG K. ENDERS

Blimp 3 User's Guide

Brian T. Keller & Craig K. Enders

Updated 10.28.2024

The Blimp software is available for download at www.appliedmissingdata.com. Blimp was developed by Craig Enders, Brian Keller, Han Du, and Roy Levy with funding from Institute of Educational Sciences awards R305D150056 and R305D190002.

Craig K. Enders, P.I. Email: cenders@ucla.edu.
Brian T. Keller, Co-P.I. Email: btkeller@missouri.edu
Han Du, Co-P.I. Email: hdu@psych.ucla.edu
Roy Levy, Co-P.I. Email: Roy.Levy@asu.edu

Programming and graphical interface by Brian T. Keller

Cite as: Keller, B. T., & Enders, C. K. (2023). Blimp user's guide (Version 3). Retrieved from www.appliedmissingdata.com/blimp

DISCLAIMER: This is free software with no expressed license given. There is no right to distribute the software or documentation without written consent. This is for your sole use only, given as is.

Table of Contents

Table of Contents.....	3
-------------------------------	----------

1 Introduction.....	7
1.1 Blimp Overview.....	7
1.2 Working in Blimp Studio.....	8
1.3 rblimp: Blimp in R.....	12
1.4 Blimp's Modeling Framework.....	14
1.5 Specifying Models for Incomplete Predictors.....	16
1.6 Missing Data Handling.....	22
1.7 Prior Distributions.....	24
1.8 New Features.....	26
1.9 Running From Terminal.....	28
2 Blimp Command Language.....	30
2.1 Overview.....	30
2.2 Blimp Commands.....	33
DATA Command.....	33
VARIABLE Command.....	34
ORDINAL Command.....	34
NOMINAL Command.....	35
COUNT Command.....	35
CLUSTERID Command.....	36
WEIGHT Command.....	36
MISSING Command.....	37
LATENT Command.....	37
RANDOM EFFECT Command.....	38
TRANSFORM Command.....	39
BYGROUP Command.....	43
FIXED Command.....	44
CENTER Command.....	44
MODEL Command.....	46
Regression Models.....	46
Discrete Outcomes.....	50
Discrete Predictors.....	50
Interaction and Polynomial Terms.....	52
Correlations and Residual Correlations.....	54
Parameter Constraints.....	56
Auxiliary Variables.....	57
Latent Variables.....	57
Multilevel Regression Models.....	59

Functions Embedded in Equations.....	64
SIMPLE Command.....	68
PARAMETERS Command.....	70
WALDTEST Command.....	73
FCS Command.....	76
BURN Command.....	79
ITER Command.....	80
CHAINS Command.....	80
NIMPS Command.....	81
THIN Command.....	81
OPTIONS Command.....	82
OUTPUT Command.....	83
SAVE Command.....	85
3 Diagnosing Convergence and Specifying the Number of Iterations.....	88
4 Regression Model Analysis Examples.....	95
4.1: Correlations and Descriptive Statistics.....	96
4.2: Polychoric Correlations With Latent Response Variables.....	98
4.3: Linear Regression.....	99
4.4: Model-Based Multiple Imputation.....	100
4.5: Linear Regression With Nominal Predictors.....	102
4.6: Fully Conditional Specification Multiple Imputation.....	105
4.7: Auxiliary Variables.....	107
4.8: Moderated Regression With an Interaction.....	110
4.9: Multiple Imputation Within Subgroups.....	114
4.10: Curvilinear Regression.....	116
4.11: Probit Regression With a Binary Outcome.....	118
4.12: Probit Regression With an Ordinal Outcome.....	120
4.13: Logistic Regression With a Binary Outcome.....	122
4.14: Logistic Regression With a Multicategorical Outcome.....	125
4.15: Count Regression.....	126
4.16: Zero-Inflated Count Outcome.....	128
4.17: Scale Scores With Incomplete Item Responses.....	131
4.18: Scale Score Interactions.....	134
4.19: Skewed Predictor With a Yeo-Johnson Transformation.....	136
4.20: Skewed Outcome With a Yeo-Johnson Transformation.....	138
4.21: Propensity Score Estimation With Missing Data.....	143
4.22: Sampling Weights.....	146

4.23: Wald Significance Tests.....	147
5 Path Analysis and Latent Variable Model Examples.....	151
5.1: Mediation Analysis.....	152
5.2: Moderated Mediation.....	154
5.3: Mediation With a Binary Outcome.....	156
5.4: Mediation With a Categorical Mediator.....	161
5.5: Mediation With a Count Outcome.....	165
5.6: Mediation With a Zero-Inflated Count Outcome.....	168
5.7: CFA With Continuous Indicators.....	171
5.8: CFA With Binary Indicators (Two-Parameter IRT Model).....	173
5.9: CFA With Ordinal Indicators.....	177
5.10: Imputing Latent Response Scores for Item-Level Factor Analysis.....	180
5.11: Skewed Indicators With a Yeo-Johnson Transformation.....	183
5.12: Latent Variable Regression Model.....	189
5.13: Latent-by-Manifest Variable Interaction.....	191
5.14: Moderated Nonlinear Factor Analysis (MNLFA).....	194
5.15: Latent-by-Latent Variable Interaction.....	198
5.16: Three-Way Latent Variable Interaction.....	201
5.17: Latent Growth Curve Model.....	203
5.18: Residual SEM: AR1 Growth Model.....	206
5.19: Random Intercept Cross-Lagged Panel Model (RI-CLPM).....	211
6 Multilevel Model Analysis Examples.....	224
6.1: Random Intercept Model.....	225
6.2: Two-Level Fully Conditional Specification Multiple Imputation.....	226
6.3: Random Coefficient Model.....	229
6.4: Multilevel SEM With Random Coefficients.....	233
6.5: Alternate Prior Distributions for Random Effect Covariance Matrix.....	236
6.6: Inspecting Residuals.....	244
6.7: Heterogeneous Within-Cluster Variation.....	247
6.8: Location-Scale Model With Heterogeneous Within-Cluster Variation.....	250
6.9: Random Effects Predicting a Level-2 Outcome.....	254
6.10: Latent Contextual Effect Model.....	257
6.11: Cross-Level Interaction Effect.....	259
6.12: 1-1-1 Mediation With Random Slopes.....	262
6.13: 1-1-1 Moderated Mediation.....	265
6.14: Within- and Between-Level Mediation.....	268
6.15: Two-Level Growth Model.....	272

6.16: Three-Level Growth Model.....	273
6.17: Three-Level SEM Growth Model.....	276
6.18: Two-Level MIMIC Measurement Model.....	279
6.19: Sampling Weights.....	281
6.20: Partially Nested Design (Singleton Clusters).....	283
6.21: Discrete-Time Survival Model.....	284
7 Missing Not at Random Process Analysis Examples.....	289
7.1: Selection Model for Linear Regression.....	289
7.2: Pattern Mixture Model for Linear Regression.....	293
7.3: Shared Parameter (Wu–Carroll) Latent Curve Model.....	302
7.4: Diggle-Kenward Latent Curve Model.....	305
7.5: Factor Analysis With a Selection Model.....	309
7.6: Mediation Analysis With a Selection Model.....	313
7.7: Two-Level Hedeker-Gibbons Pattern Mixture Growth Model.....	316
7.8: Selection Model for a Two-Level Regression With Random Coefficients.....	319
7.9: Two-Level Shared Parameter (Wu–Carroll) Growth Curve Model.....	321
8 Monte Carlo Studies in Blimp.....	324
9 References.....	326

1 Introduction

1.1 Blimp Overview

Blimp is an all-purpose data analysis and latent variable modeling program that harnesses the flexible power of Bayesian estimation in a user-friendly application that requires minimal scripting and no deep-level knowledge about Bayes. The application, which is available for macOS, Windows, and Linux, was developed with funding from Institute of Educational Sciences awards R305D150056 and R305D190002. The application began as a platform for implementing multilevel multiple imputation via fully conditional specification (Enders, Keller, & Levy, 2018), and its second release transitioned the software to a full-featured multilevel analysis package (Enders, Du, & Keller, 2020). Blimp 3 introduces wide ranging and powerful capabilities for multivariate analyses with latent variables (e.g., path models, measurement models, structural equation models), including many models not available in other software packages.

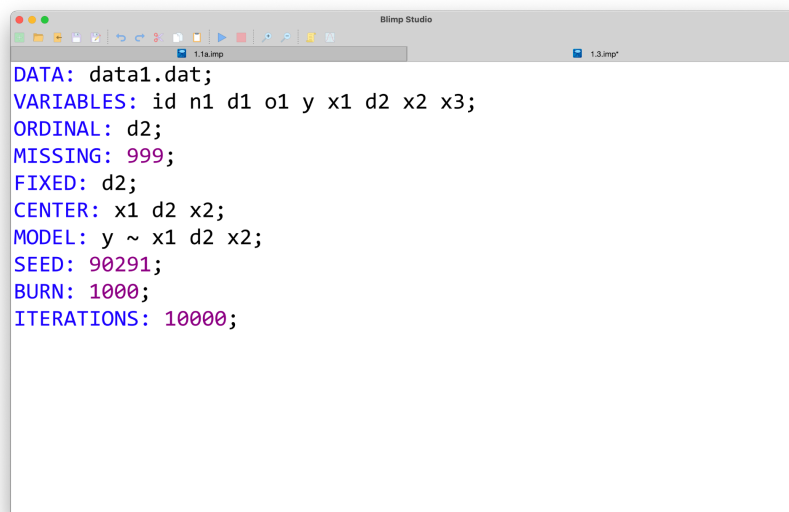
The development team's philosophy for Blimp is to bring easy Bayes estimation to the masses; the program offers some opportunities for "getting under the hood", but algorithmic tweaks and nuanced model specifications are not as customizable as they are in specialized (but less user-friendly) programs such as Stan or JAGS. To this end, Blimp implements a reasonable set of diffuse or noninformative prior distributions with a handful of "off-the-shelf" alternatives described in Bayesian texts. In line with our overarching philosophy, complex models can be specified with minimal coding by simply listing variable names in a format that resembles a regression equation (e.g., $y \sim x_1 \ x_2 \ x_3$). In most cases, Blimp automatically introduces means, variances, and covariances (or correlations) with no additional specifications required, and the software also adds any supporting models needed for missing data handling.

Blimp's primary purpose is to provide researchers with a powerful tool for analyzing data, with or without missing values. Blimp 3 offers a commercial-grade user

experience with the flexibility to estimate complex latent variable models, many of which are not available in other software packages. Models can include up to three levels with mixtures of binary, ordinal, multicategorical nominal, normal, skewed continuous variables, and count variables. Chapters 4 through 7 of this guide provides numerous examples. Separate from its data analytic core, Blimp continues to offer the fully conditional specification routines introduced in Version 1. Blimp's implementation of fully conditional specification parallels van Buuren's popular MICE program (van Buuren & Groothuis-Oudshoorn, 2011), but it uses latent response variable framework to treat categorical variables and uses latent group means to preserve multilevel data structures. Enders (2022) describes fully conditional specification with latent variables, and Chapters 4 through 6 of this guide provides illustrations.

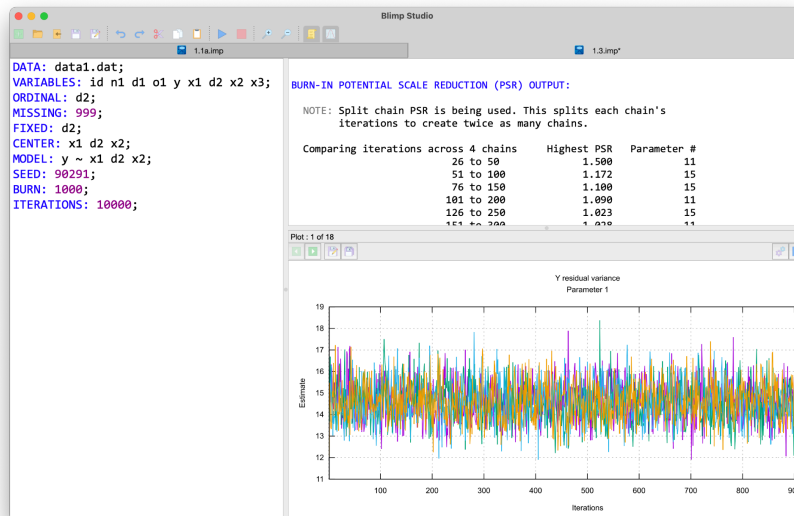
1.2 Working in Blimp Studio

One of the major features in Blimp 3 is a redesigned graphical user interface called Blimp Studio. The Studio application features a tabbed interface that makes it easy to work with multiple scripts and projects at the same time. The graphic below shows the Blimp Studio interface.

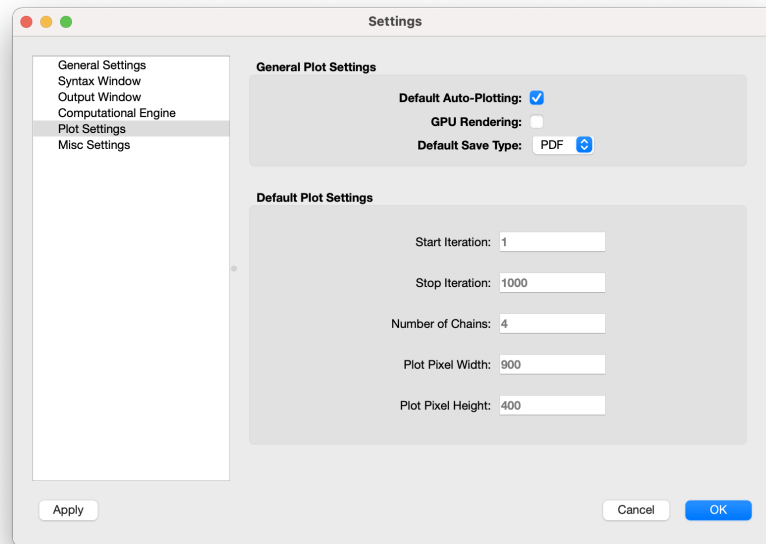
A screenshot of the Blimp Studio application window. The window title is "Blimp Studio" and it has a tab labeled "1.3.imp". The main area contains a script with the following content:

```
DATA: data1.dat;  
VARIABLES: id n1 d1 o1 y x1 d2 x2 x3;  
ORDINAL: d2;  
MISSING: 999;  
FIXED: d2;  
CENTER: x1 d2 x2;  
MODEL: y ~ x1 d2 x2;  
SEED: 90291;  
BURN: 1000;  
ITERATIONS: 10000;
```


Clicking the blue arrow button on the toolbar executes a script and spawns a paned interface that adds an output window containing the analysis results and a plotting window displaying trace (time series) plots for every model parameter.



Clicking on the normal distribution icon in the toolbar hides the plotting window, which can also be disabled completely in the application's Preferences, located under the **Blimp Studio > Preferences** pull-down menu. Other visual settings such as fonts and the orientation of the paned windows can also be set in the Preferences pane, shown below.



The MCMC summary tables on Blimp's output include unstandardized coefficients, standardized slopes, and variance explained effect size estimates (Rights & Sterba, 2019). Certain types of analyses produce additional output (e.g., odds ratios in a logistic regression; transformation parameters for skewed variables). The graphic below shows a typical tabular display of the analysis results from a regression model. Blimp automatically saves an output file with a `.blimp-out` extension to the same directory as the analysis script. The outputs are linked to their analysis scripts, such that double-clicking on one of the files opens both in the Blimp Studio interface.

The screenshot shows the Blimp Studio interface with a code editor on the left and a results pane on the right. The code editor contains the following text:

```
DATA: data1.dat;
VARIABLES: id v1 v2 v3 y x1 d x2 v4;
ORDINAL: d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL: y ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITERATIONS: 10000;
```

The results pane displays the following information:

OUTCOME MODEL ESTIMATES:
Summaries based on 10000 iterations using 2 chains.

Outcome Variable: *y*
Grand Mean Centered: x1 x2

Parameters	Median	StdDev	2.5%	97.5%	ChiSq	PValue	N_Eff
Variances:							
Residual Var.	14.611	0.928	12.916	16.584	---	---	6086.359
Coefficients:							
Intercept	27.701	0.256	27.208	28.204	11669.256	0.000	4675.371
x1	0.588	0.060	0.472	0.706	96.354	0.000	5119.163
x2	0.191	0.045	0.101	0.277	18.167	0.000	4315.672
d	1.807	0.343	1.130	2.479	27.697	0.000	6642.630
Standardized Coefficients:							
x1	0.401	0.036	0.328	0.468	124.838	0.000	5002.048
x2	0.173	0.040	0.092	0.249	18.824	0.000	4367.722
d	0.204	0.038	0.128	0.276	29.208	0.000	6503.835
Proportion Variance Explained							
by Coefficients	0.259	0.033	0.195	0.323	---	---	4367.193
by Residual Variation	0.741	0.033	0.677	0.805	---	---	4367.193

PREDICTOR MODEL ESTIMATES:
Summaries based on 10000 iterations using 2 chains.

MCMC estimation produces a distribution for each model parameter. The median and standard deviation columns describe the center and spread of the posterior distributions; although they make no reference to drawing repeated samples, they are analogous—and numerically equivalent in most cases—to frequentist point estimates and standard errors. The 95% credible intervals in the rightmost columns give a range that captures 95% of the parameter's distribution. These are akin to confidence intervals, but the intervals describe parameter distributions rather than characteristics of repeated samples.

Although MCMC estimation is grounded in the Bayesian statistical paradigm, one can also view posterior medians, standard deviations, and credible intervals as surrogates for frequentist point estimates, standard errors, and confidence intervals. Levy and McNeish (2023) describe this perspective as “computational frequentism”. Essentially, the researcher wants to operate within the frequentist framework, but they use MCMC to solve a difficult estimation problem. Missing data analyses are a

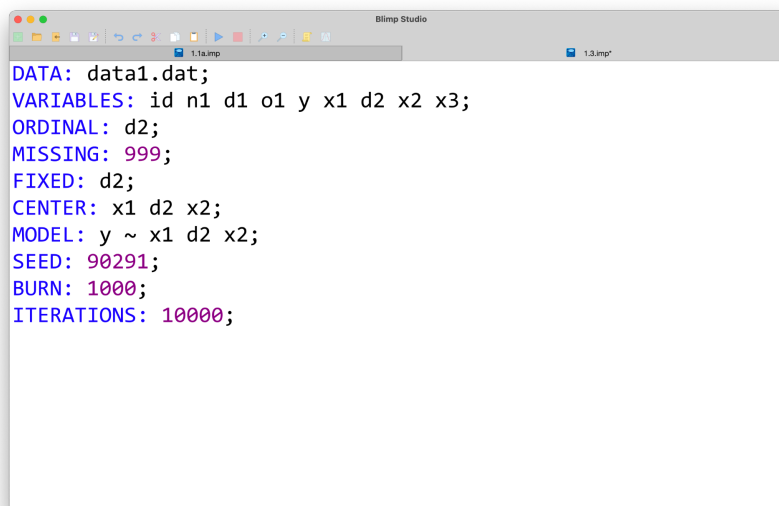
compelling use case for computational frequentism because optimal likelihood-based solutions are not always available or easy to use. To facilitate this perspective, the Blimp output also includes a chi-square statistic and p-value for each model parameter (the Bayesian Wald test; Asparouhov & Muthén, 2021). These Wald tests are like squared z-statistics from maximum likelihood estimation, but MCMC generates the point estimate and “standard error” for the test.

1.3 rblimp: Blimp in R

The `rblimp` package for R is currently available for download at [Brian Keller's github](#). The standalone version of Blimp must be installed prior to downloading and installing `rblimp`. The package is accessed using the `remotes` package. Executing the following command downloads and installs `rblimp`. A zip archive of the `rblimp` scripts for the analysis examples in Chapters 4 through 7 is [here](#).

```
# install.packages('remotes')
remotes::install_github('blimp-stats/rblimp')
```

When working in R, each Blimp command is simply an input parameter into the `rblimp` function. To illustrate, consider the following script from the GUI.



```
DATA: data1.dat;
VARIABLES: id n1 d1 o1 y x1 d2 x2 x3;
ORDINAL: d2;
MISSING: 999;
FIXED: d2;
CENTER: x1 d2 x2;
MODEL: y ~ x1 d2 x2;
SEED: 90291;
BURN: 1000;
ITERATIONS: 10000;
```

The corresponding R script is as follows.

```
library(rblimp)

setwd('~/.desktop/')
data1 <- as.data.frame(read.table('data1.dat', na.strings = '999'))
colnames(data1) <- c('id', 'v1', 'd1', 'o1', 'y', 'x1', 'd2', 'x2', 'x3')

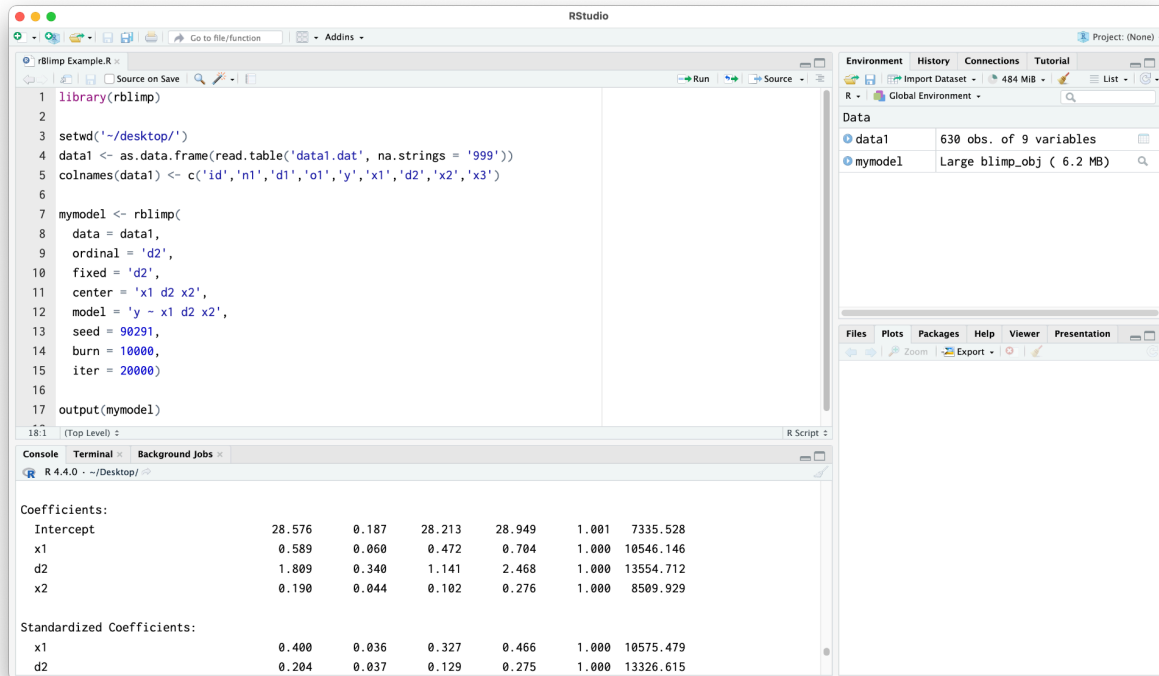
mymodel <- rblimp(
  data = data1,
  ordinal = 'd2',
  fixed = 'd2',
  center = 'x1 d2 x2',
  model = 'y ~ x1 d2 x2',
  seed = 90291,
  burn = 10000,
  iterations = 20000
)
output(mymodel)
```

Each command in the Blimp script (each capitalized word) is an input parameter in the `rblimp` function. The two exceptions are the `VARIABLES` and `MISSING` commands, which are omitted because that information is contained in the R data file. Following R convention, the input parameters are separated by commas. Alphanumeric inputs like model statements, variable lists, transformations, and new parameters are enclosed in quotes. Numeric inputs like the seed and number of iterations do not require quotes. Finally, subcommands that are part of the same command (e.g., different equations in the `MODEL` command) are separated by semicolons, as they are in the Blimp script.

```
model = '
  d2 ~ x2;
  x1 ~ d2 x2;
  y ~ x1 d2 x2;'
```

The standalone version of Blimp must be installed prior to downloading and installing `rblimp` because the R function calls the Blimp computational engine from

the applications folder. Blimp otherwise behaves like any other package in the R environment.



```

1 library(rblimp)
2
3 setwd("~/desktop/")
4 data1 <- as.data.frame(read.table("data1.dat", na.strings = '999'))
5 colnames(data1) <- c('id','n1','d1','o1','y','x1','d2','x2','x3')
6
7 mymodel <- rblimp(
8   data = data1,
9   ordinal = 'd2',
10  fixed = 'd2',
11  center = 'x1 d2 x2',
12  model = 'y ~ x1 d2 x2',
13  seed = 90291,
14  burn = 10000,
15  iter = 20000)
16
17 output(mymodel)

```

Console Output:

```

R 4.4.0 - ~/Desktop/

Coefficients:
Intercept          28.576    0.187    28.213    28.949    1.001    7335.528
x1                 0.589    0.060    0.472    0.704    1.000   10546.146
d2                 1.809    0.340    1.141    2.468    1.000   13554.712
x2                 0.190    0.044    0.102    0.276    1.000    8509.929

Standardized Coefficients:
x1                 0.400    0.036    0.327    0.466    1.000   10575.479
d2                 0.204    0.037    0.129    0.275    1.000   13326.615

```

All results that Blimp produces (estimates, MCMC samples, imputations, diagnostics, etc.) are automatically saved in the `rblimp` model object. Saved quantities can be inspected by typing `@` after the model object's name in the R console. The analysis examples in Chapters 4 through 7 include both the Blimp Studio and `rblimp` scripts.

1.4 Blimp's Modeling Framework

The major feature that distinguishes Blimp's estimation architecture from other latent variable modeling software packages is that it does not work with the joint distribution of the analysis variables. Rather, the multivariate distribution is factored into the product of multiple univariate distributions. To illustrate, consider an analysis involving Y , X , and M . The trivariate distribution factors into the product of three univariate distributions, each of which corresponds to a univariate regression model.

$$f(Y, X, M) = f(Y|X, M) \times f(X|M) \times f(M)$$

Blimp estimates the models on the right of the equals sign without assuming anything about the form or shape of the multivariate distribution on the left. The advantage of this specification is that the individual regression equations can feature mixtures of categorical and normal variables, continuous variables with skewed distributions, interactive or nonlinear terms, and other complex constructions. In such cases, the multivariate distribution on the left doesn't have a known or simple form, and model misspecifications (e.g., treating such data as multivariate normal) can introduce bias.

The theory for Blimp's model specification is given by Ibrahim and colleagues (Ibrahim, Chen, & Lipsitz, 2002; Ibrahim, Lipsitz, & Chen, 1999; Lipsitz & Ibrahim, 1996), and the software extends these ideas to latent variable models with up to three levels. More recent literature refers to this model specification as fully Bayesian estimation, the sequential specification, and factored regression (Enders et al., 2020; Erler, Rizopoulos, Jaddoe, Franco, & Lesaffre, 2019; Erler et al., 2016; Lüdtke, Robitzsch, & West, 2020a, 2020b; Zhang & Wang, 2017).

To illustrate Blimp's modeling framework more concretely, consider a research scenario where the focal analysis model is a linear regression of Y on X and M . The factorization above translates into the following linear regression models, where all residuals are normal and have constant variance.

$$Y = \beta_0 + \beta_1 X_i + \beta_2 M_i + \varepsilon_i$$

$$X_i = \gamma_{01} + \gamma_{11} M_i + r_{1i}$$

$$M_i = \gamma_{02} + r_{2i}$$

The X and M equations are essentially nuisance models in this example, and their role is to link incomplete predictors to one another as well as to any complete regressors.

Any univariate equation can feature mixtures of categorical and normal variables, continuous variables with skewed distributions, and interactive or nonlinear terms, among other things. For example, the following equations include an interaction between X and M in the focal model and a quadratic association between X and M in the supporting predictor model.

$$Y = \beta_0 + \beta_1 X + \beta_2 M + \beta_3 (X)(M) + \varepsilon$$

$$X = \gamma_{01} + \gamma_{11} M + \gamma_{21} M^2 + r_1$$

$$M = \gamma_{02} + r_2$$

If either X or M has missing values, a joint modeling framework is inappropriate and would produce biased estimates because the incomplete predictor distributions are complicated nonlinear functions of the outcome; such associations are fundamentally incompatible with off-the-shelf distributions such as the multivariate normal (Bartlett, Seaman, White, & Carpenter, 2015; Liu, Gelman, Hill, Su, & Kropko, 2014). Specifying a model as a sequence of factored regressions bypasses the problematic joint distribution altogether. These ideas readily extend to latent variables, which Blimp views as missing data to be estimated (imputed).

1.5 Specifying Models for Incomplete Predictors

Throughout the guide, we use the term “predictors” to refer to exogenous variables—in a path diagram, variables that do not have incoming arrows. When predictors are complete, there is usually no reason to specify a distribution for these variables. Instead, the covariate data essentially function as known constants, as in ordinary least squares. In contrast, incomplete predictors require an explicit distribution for imputation. Blimp allows these distributions to have many forms (e.g., normal, skewed, discrete). In most cases, assigning a distribution to a predictor means making that regressor a dependent variable in its own regression model. These supporting models can be explicitly specified, or Blimp can create them automatically. These two strategies are somewhat different and have different

strengths and weaknesses. We use the following multiple regression to illustrate the two model specification strategies, and we assume that all variables are incomplete.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \varepsilon$$

To begin, consider the situation where Blimp automatically constructs supporting regression models for the predictors. The `MODEL` statement is as follows.

MODEL:

```
y ~ x1 x2 x3;
```

Throughout the guide, we refer to this specification as reflecting **unspecified associations** among the predictors. The underlying regression models follow a round robin pattern where each predictor is regressed on all other predictors.

$$X_1 = \mu_1 + \gamma_{11}(X_2 - \mu_2) + \gamma_{21}(X_3 - \mu_3) + r_1$$

$$X_2 = \mu_2 + \gamma_{12}(X_1 - \mu_1) + \gamma_{22}(X_3 - \mu_3) + r_2$$

$$X_3 = \mu_3 + \gamma_{13}(X_1 - \mu_1) + \gamma_{23}(X_2 - \mu_2) + r_3$$

The regressions above are linear and assume normally distributed residuals, but this specification also allows for binary, ordinal, and multicategorical nominal predictors, in which case Blimp adopts a latent response variable formulation (Albert & Chib, 1993; Carpenter & Kenward, 2013; Enders et al., 2018; Johnson & Albert, 1999). Variable metrics are specified using the `ORDINAL` and `NOMINAL` commands described in Chapter 2. Enders et al. (2020) describe the multilevel version of this specification.

More formally, adopting unspecified associations for the predictors invokes a model that factors the joint distribution into the product of a univariate distribution for the analysis model and a multivariate distribution for the predictors.

$$f(Y, X_1, X_2, X_3) = f(Y|X_1, X_2, X_3) \times f(X_1, X_2, X_3)$$

We refer to this setup as a **partially factored specification** because it leaves the rightmost term—a multivariate normal distribution for continuous predictors and

latent response variables—unfactored. With mixed response types, the multivariate distribution's covariance matrix is difficult to model because it could include a mixture of fixed constants, variances and covariances, and correlations. The round robin regression equations above simplify estimation by leveraging the property that a multivariate normal distribution spawns an equivalent set of linear regression models (Arnold, Castillo, & Sarabia, 2001; Liu, Gelman, Hill, Su, & Kropko, 2014). Importantly, the normal distribution assumption precludes the possibility of nonlinear associations among the predictors, as such relations are incompatible with normal data.

Next, consider the situation where the user explicitly specifies the regression equations for the predictors. This specification leverages the probability chain rule to factorize the joint distribution of the analysis variables into the product of several univariate conditional distributions, each of which corresponds to a regression model.

$$f(Y, X_1, X_2, X_3) = f(Y|X_1, X_2, X_3) \times f(X_1|X_2, X_3) \times f(X_2|X_3) \times f(X_3)$$

The corresponding regression equations follow the same cascading pattern where the first predictor's model is empty, the second predictor is regressed on the first, the third on the first and second, and so on.

$$X_1 = \gamma_{01} + \gamma_{11}X_2 + \gamma_{21}X_3 + r_1$$

$$X_2 = \gamma_{02} + \gamma_{12}X_3 + r_2$$

$$X_3 = \gamma_{03} + r_3$$

The **MODEL** statement for this specification is

```
MODEL:
predictor.models:
x3 ~ 1;
x2 ~ x3;
```

```
x1 ~ x2 x3;
focal.model:
y ~ x1 x2 x3;
```

and the code block below illustrates a syntax shortcut for this specification that lists all predictors to the left of a tilde.

```
MODEL:
predictor.models:
x1 x2 x3 ~ 1;
focal.model:
y ~ x1 x2 x3;
```

We refer to this setup as a **factored regression specification** or **sequential specification** (Erler et al., 2016; Lüdtke et al., 2020b).

Blimp's output produces a tabular summary for each estimated model (there are four in the previous example). By default, the summary tables are alphabetized according to the outcome's name. Some users may prefer to order the tables so the primary analysis results appear first. This is accomplished by using labels ending in a colon to define blocks of models that appear in the order listed. The following example creates two model blocks, such that the focal model summary table appears first on the printed output.

```
MODEL:
focal.model:
y ~ x1 x2 x3;
predictor.models:
x1 x2 x3 ~ 1;
```

The sequential specification for predictors differs in important ways. First, the predictor's equation can have any metric allowed by Blimp—normal, skewed continuous, binary (probit or logit link), ordinal (probit link), multicategorical nominal (logistic link), or count (negative binomial link). Second, associations among the

predictors need not be linear. For example, the following equations include the quadratic effect of X_3 on X_2 .

$$X_1 = \gamma_{01} + \gamma_{11}X_2 + \gamma_{21}X_3 + r_1$$

$$X_2 = \gamma_{02} + \gamma_{12}X_3 + \gamma_{12}X_3^2 + r_2$$

$$X_3 = \gamma_{03} + r_3$$

The corresponding MODEL statement is as follows.

MODEL:

predictor.models:

x3 ~ 1;

x2 ~ x3 (x3^2);

x1 ~ x2 x3;

focal.model:

y ~ x1 x2 x3;

When using a sequential specification, ordering the predictors in a particular way can facilitate estimation and reduce the impact of model misspecifications. Lüdtke et al. (2020b, pp. 171-172) recommend ordering variables from left to right by their missingness rates, with categorical variables before continuous variables. To illustrate, suppose that X_1 is an incomplete binary variable, X_2 is complete, and X_3 is an incomplete continuous variable. Their recommended specification would be as follows

$$f(Y|X_1, X_2, X_3) \times f(X_3|X_1, X_2) \times f(X_1|X_2) \times f(X_2)$$

and the corresponding model specification is

ORDINAL: x1;

MODEL:

predictor.models:

x2 ~ 1;

x1 ~ x2;

x3 ~ x1 x2;

focal.model:

```
y ~ x1 x2 x3;
```

or simply as follows.

```
ORDINAL: x1;
MODEL:
predictor.models:
x3 x1 x2 ~ 1;
focal.model:
y ~ x1 x2 x3;
```

Finally, when predictors are complete, there is usually no reason to specify a distribution for these variables; instead, the covariate data essentially function as known constants, as in ordinary least squares. With either specification for the predictors, listing complete predictors on the **FIXED** command line indicates that the variable does not require a model (or distribution). Continuing with the previous example where X_2 is complete, the sequential specification moves the complete variable from the left to the right of the tilde, as follows.

```
FIXED: x2;
MODEL:
predictor.models:
x3 x1 ~ x2;
focal.model:
y ~ x1 x2 x3;
```

The partially factored specification with unspecified predictor associations is as follows.

```
FIXED: x2;
MODEL:
y ~ x1 x2 x3;
```

The examples in Chapters 4 through 7 generally treat predictor distributions as unspecified. This setup is easy to specify, and it is also convenient for centering because the means are explicit model parameters that MCMC iteratively estimates.

This approach does not limit the composition of the focal analysis model, which can include interactive or nonlinear terms. However, predictor regressions are necessarily additive, as interactions and similar nonlinearities are incompatible with a multivariate normal distribution. As mentioned previously, unspecified predictor associations accommodate normal, binary, ordinal, and multicategorical variables via a latent response variable framework. Blimp can apply a Yeo-Johnson (Yeo & Johnson, 2000) distribution to skewed variables and negative binomial regression to count variables, but these features require a sequential specification.

The next section provides a complete description of the Blimp command language, and Chapters 4 through 7 provide numerous analysis examples. The examples span a wide variety of single-level and multilevel analyses with manifest and latent variables, including analyses for missing not at random processes.

1.6 Missing Data Handling

As detailed in Section 1.3, Blimp's estimation architecture factorizes a multivariate distribution into the product of univariate distributions. This factorization carries through to missing data handling, where the distributions of missing values rely on a collection of univariate models. The advantage of this specification is that Blimp can generate appropriate imputations from models that are fundamentally incompatible with known multivariate distributions. Examples include models with incomplete interactive or polynomial effects, multilevel models with random effects, and models with skewed variables or mixtures of discrete and numeric variables.

To illustrate missing data handling, consider an analysis involving Y , X , and M . To refresh, the trivariate distribution factors into the product of three univariate distributions, each of which corresponds to a regression model.

$$f(Y, X, M) = f(Y|X, M) \times f(X|M) \times f(M)$$

Blimp estimates the models on the right of the equals sign without assuming anything about the form or shape of the multivariate distribution on the left. In a

simple scenario where all three variables are continuous, the factorization corresponds to the following linear regression models.

$$Y = \beta_0 + \beta_1 X_i + \beta_2 M_i + \varepsilon_i$$

$$X_i = \gamma_{01} + \gamma_{11} M_i + r_{1i}$$

$$M_i = \gamma_{02} + r_{2i}$$

In a factored regression framework, the distributions of missing values depend on every model in which a variable appears. For example, the distribution of missing Y values depends only on the analysis model, and MCMC samples imputations from a normal curve with center and spread equal to a predicted value and residual variance, respectively.

$$f(Y|X, M) = N(\beta_0 + \beta_1 X_i + \beta_2 M_i, \sigma_\varepsilon^2)$$

Because it appears in two models—once as a predictor and once as an outcome—the conditional distribution of missing X values is proportional to the product of two normal distributions.

$$\begin{aligned} f(X|Y, M) &\propto f(Y|X, M) \times f(X|M) = \\ &N(\beta_0 + \beta_1 X_i + \beta_2 M_i, \sigma_\varepsilon^2) \times N(\gamma_{01} + \gamma_{11} M_i, \sigma_{r1}^2) \end{aligned}$$

In a similar vein, the conditional distribution of the missing M values is proportional to the product of three normal distributions.

$$f(M|Y, X) \propto N(\beta_0 + \beta_1 X_i + \beta_2 M_i, \sigma_\varepsilon^2) \times N(\gamma_{01} + \gamma_{11} M_i, \sigma_{r1}^2) \times N(\gamma_{02}, \sigma_{r2}^2)$$

These conditional distributions have analytic solutions in many cases (Levy & Enders, 2021), but Blimp's MCMC algorithm uses Metropolis sampling to draw imputations from composite functions such as these.

With a collection of additive models like those above, the distributions of missing values are equivalent to the distributions implied by a joint modeling framework or

fully conditional specification. The same is not true for models with nonlinearities, skewed variables, or mixtures of discrete and numeric variables. To illustrate, suppose the analysis model includes an interaction between X and M . The factorization and the corresponding regression models are as follows.

$$f(Y, X, M) \propto f(Y|X, M, X \times M) \times f(X|M) \times f(M)$$

$$Y = \beta_0 + \beta_1 X_i + \beta_2 M_i + \beta_3 (X_i \times M_i) + \varepsilon$$

$$X_i = \gamma_{01} + \gamma_{11} M_i + r_{1i}$$

$$M_i = \gamma_{02} + r_{2i}$$

As before, the distributions of missing values depend on every model in which a variable appears. For example, the distribution of missing X values is again the product of two normal distributions, as follows.

$$f(X|Y, M) \propto f(Y|X, M, X \times M) \times f(X|M) =$$

$$N(\beta_0 + \beta_1 X_i + \beta_2 M_i + \beta_3 (X_i \times M_i), \sigma_\varepsilon^2) \times N(\gamma_{01} + \gamma_{11} M_i, \sigma_{r1}^2)$$

Importantly, the conditional distribution of missing values is incompatible with multivariate normality because its variance is heteroscedastic function (Enders et al., 2020, Eq. 8). The same issue applies more broadly to models with polynomial or nonlinear terms and multilevel models with random effects, among others.

Basing imputations on factored regression specification is guaranteed to produce a set of compatible univariate regressions, whereas conventional modeling frameworks that create imputations based on a multivariate distribution are prone to bias (Bartlett, Seaman, White, & Carpenter, 2015; Liu, Gelman, Hill, Su, & Kropko, 2014). More generally, the univariate models described above could feature discrete variables (binary, ordinal, multicategorical nominal, count), skewed continuous variables, and even latent variables, which Blimp views as missing data to be estimated (imputed).

1.7 Prior Distributions

Blimp's MCMC algorithm adopts different prior distributions that depend on the variable's metric and the type of model. Blimp invokes a normal distribution prior for regression coefficients (and means). We denote these priors as `normal(mu, var)`. The default noninformative prior is a normal distribution with infinite variance. Variances and residual variances use a conjugate inverse gamma prior distribution. The inverse gamma distribution has two parameters, scale and shape. We denote this distribution as `invgamma(a, b)`. The scale parameter can be viewed as the prior degrees of freedom $\div 2$, and the shape parameter is the prior sums of squares $\div 2$. For covariance matrices, Blimp adopts conjugate inverse Wishart priors. The inverse Wishart distribution is the multivariate extension of the inverse gamma. It too has two parameters, a scale matrix and degrees of freedom. We denote this prior as `IW(SSp, dfp)`, where `SSp` is the prior sum of squares and cross-products matrix (the scale matrix), and `dfp` is the prior degrees of freedom. In the table below, we use p to denote the number of variance in the covariance matrix. The table below summarizes the default prior distributions, and the rightmost column indicates whether the user can specify a custom prior. Priors are modified using either the `OPTIONS` command or the `PARAMETERS` command.

Variable Type	Parameter	Default Prior	Modifiable
Continuous exogenous manifest predictors with unspecified associations	Regression coefficients	<code>normal(0, Inf)</code>	No
Continuous exogenous manifest predictors with unspecified associations	Residual variance	<code>invgamma(1, .5)</code>	Yes (<code>OPTIONS</code> command)
Ordinal exogenous manifest predictor (> two categories)	Threshold	<code>uniform(-Inf, Inf)</code>	No
Discrete exogenous manifest predictor	Residual variance	NA (fixed parameter)	NA

Continuous outcome (manifest or latent)	Regression coefficients	$\text{normal}(\theta, \text{Inf})$	Yes (PARAMETERS command)
Continuous outcome (manifest or latent)	Residual variance	$\text{invgamma}(-1, \theta)$	Yes (OPTIONS and PARAMETERS commands)
Discrete outcome	Regression coefficients	$\text{normal}(\theta, 5)$	Yes (PARAMETERS command)
Discrete outcome	Residual variance	NA (fixed parameter)	NA
Ordinal outcome (> two categories)	Threshold	$\text{uniform}(-\text{Inf}, \text{Inf})$	No
Random effect residual in a mixed model	Covariance matrix	$\text{IW}(\theta, -p-1)$	Yes (OPTIONS command)
Continuous outcome (manifest, latent, or binary/ordinal latent response variables)	Covariance matrix	$\text{IW}(\theta, -p-1)$	Yes (OPTIONS command)

1.8 New Features

The following is a list of new features and functionality available in Version 3.2.

- ❖ Multiple-equation models (e.g., path models) with up to three levels
- ❖ Latent variables and latent variable regressions
- ❖ Latent variables with random effects, interactions, and nonlinear effects
- ❖ Single-level and multilevel selection and pattern mixtures models for missing not at random processes
- ❖ Multivariate regression models
- ❖ Parameter constraints
- ❖ Auxiliary parameters that are functions of estimated parameters
- ❖ Latent variable imputation
- ❖ Yeo-Johnson modeling for skewed continuous variables
- ❖ Binary and multinomial logistic regression
- ❖ Negative binomial regression for count outcomes

- ❖ Estimation with sampling weights
- ❖ Facilities for computing new variables with numerous built-in functions
- ❖ Built-in functions embedded within regression equations
- ❖ Facilities for introducing custom univariate prior distributions
- ❖ New Blimp Studio graphical user interface
- ❖ Redesigned output with numerous enhancements and additional printing options
- ❖ Better optimization and many algorithmic improvements
- ❖ Enhanced user guide with dozens of new examples and analysis scripts
- ❖ Enhanced WALDTEST command for Bayesian Wald tests in all models Blimp estimates
- ❖ DIC and WAIC information criteria for model selection
- ❖ Correlations among the residuals of all outcome variables for evaluating local sources of model misfit

The following is a list of features and functionality that were introduced in Version 2.

- ❖ WALDTEST command for Bayesian Wald tests (Asparouhov & Muthén, 2021)
- ❖ Simplified scripting language and redesigned output
- ❖ Graphical interface with automatic updates when new features become available
- ❖ Graphical engine that creates trace plots for all model parameters
- ❖ Bayesian estimation of single-level, multilevel (up to three levels), and multiple group regression models with complete or incomplete data
- ❖ Posterior summaries of all model parameters from Bayesian estimation (posterior mean, median, standard deviation, and credible interval)
- ❖ Single-level and multilevel R-squared measures (Rights & Sterba, 2019)
- ❖ Bayesian estimation for interactive and nonlinear effects with missing data
- ❖ Bayesian estimation with grand mean centering (all models) and group mean centering (two- and three-level models)
- ❖ Post-hoc probing of interaction effects with continuous or categorical moderators
- ❖ Bayesian estimation of conditional effects (simple slopes) in regression models with interaction effects

- ❖ Incomplete binary, ordinal, or nominal predictor variables
- ❖ Discrete and latent imputations for binary, ordinal, and nominal variables
- ❖ FCS or Bayesian estimation with level-2 and level-3 cluster means modeled as latent variables
- ❖ Contextual effects models with latent group means or manifest group means
- ❖ Interaction effects with latent group means
- ❖ Various algorithmic and interface enhancements (eg, random starting values, options for saving various estimates and output)

1.9 Running From Terminal

Blimp scripts can also be executed from the terminal without the graphical interface. This is useful when conducting computer simulations, for example. The most basic specification includes a file path to the Blimp executable file followed by a file path to the script to be executed. To illustrate, the following line of code executes a script located on the desktop.

```
/Applications/Blimp/blimp ~/desktop/myscript.imp
```

Similarly, the following line uses the Blimp beta engine to execute the same file.

```
/Applications/Blimp/blimp-beta ~/desktop/myscript.imp
```

Several parts of the Blimp script can be specified via command line arguments. The general form of an argument includes a double dash followed by a keyword and an input parameter. For example, the following code block uses a command line argument to specify the input data set.

```
BLIMPPATH=/Applications/Blimp/blimp  
${BLIMPPATH} ~/desktop/myscript.imp --data ~/desktop/mydata.dat
```

Note that any parameters specified as command line arguments replace the current contents of the script (e.g., the file specified on the `DATA` command is replaced by the file `~/desktop/mydata.dat`).

In addition to the input data, command line arguments include the random number seed and most quantities exported using the `SAVE` command. The code block below shows the full array of command line arguments. The backslash is the Linux command continuation character; the arguments would otherwise need to appear on a single line separated by a space.

```
/Applications/Blimp/blimp ~/desktop/myscript.imp \  
  --seed {seed value} \  
  --data {filepath to input data} \  
  --output {filepath to blimp-out output file} \  
  --stacked {filepath to stacked imputation data} \  
  --stacked0 {filepath to stacked original + imputed data} \  
  --separate {filepath to separate imputation data sets} \  
  --estimates {filepath to save estimate summary tables} \  
  --burn {filepath to save all burn-in estimates} \  
  --iterations {filepath to save all post burn-in estimates} \  
  --psr {filepath to save burn-in psr values} \  
  --waldtest {filepath to save Wald statistics} \  

```

2 Blimp Command Language

2.1 Overview

This chapter gives a detailed account of the Blimp's scripting language. Blimp commands can be entered in the Blimp Studio syntax editor or in a plain text file with `.imp` as the file extension. The code block below shows a typical script with many of Blimp's major commands.

```
DATA: data.dat;  
VARIABLES: id a1:a4 y m x1:x3 z1 z2;  
ORDINAL: x1;  
NOMINAL: x3;  
MISSING: 999;  
FIXED: x3;  
CENTER: grandmean = x1 x2;  
MODEL:  
# x1-x3 and x2-x3 interaction predicting m;  
m ~ x1 x2 x3 x2*x3;  
# m and x1-x3 predicting y;  
y ~ m x1:x3;  
SEED: 90291;  
BURN: 2000;  
ITER: 10000;
```

The Blimp command language uses the following general conventions, most of which are shown in the previous code block.

- ❖ Upper and lower case are equivalent, no case sensitivity
- ❖ Command names (e.g., `DATA`, `VARIABLES`) end in a colon
- ❖ Subcommands or specifications following a command in a semicolon
- ❖ Commands and subcommands can span multiple lines
- ❖ A colon can be used to specify a range of variables with the same prefix and suffix
- ❖ A `#` symbol indicates a comment that Blimp ignores until the end of the line

- ❖ Three symbols are needed to specify models: (a) \sim or $<-$ denotes a regression equation, (b) $\sim\sim$ or $\sim\sim$ denote variances and covariances, and (c) $->$ or $=\sim$ assigns indicators to a latent variable
- ❖ Mathematical operator symbols are $*$ for multiplication, $/$ for division, $+$ for addition, $-$ for subtraction, $^$ or $**$ to raise a variable or quantity to a power, and parentheses for specifying order of operations

When running Blimp from within the `rblimp` package in R, each of the major commands becomes an input parameter for the function. With the exception of numeric inputs like the number of iterations, each input is enclosed in quotes as a text string. For commands that require multiple lines (e.g., models with multiple equations), multiple lines separated by the line terminator (`;`) are enclosed in quotes. Returning to the previous example, the corresponding `rblimp` script is shown below.

```
library(rblimp)

setwd('~/.desktop/')
data1 <- as.data.frame(read.table('data1.dat', na.strings = '999'))
colnames(data1) <- c('id', 'v1', 'd1', 'o1', 'y', 'x1', 'd2', 'x2', 'x3')

mymodel <- rblimp(
  data = data1,
  ordinal = 'd2',
  nominal = 'x3',
  fixed = 'd2',
  center = 'grandmean = x1 x2',
  model = '
    m ~ x1 x2 x3 x2*x3;
    y ~ m x1:x3;',
  seed = 90291,
  burn = 2000,
  iterations = 10000
)
output(mymodel)
```

Blimp also provides a number of built-in functions that work in conjunction with certain commands. The `TRANSFORM` command can use these functions to create new variables, the `PARAMETERS` command can use these routines to compute auxiliary parameters that are functions of the estimated model parameters, and functions can

be embedded within regression equations listed in the **MODEL** statement. The list of functions is below.

- ❖ **abs(x)** = absolute value of x
- ❖ **sqrt(x)** = square root of x
- ❖ **exp(x)** = exponential function applied to x
- ❖ **logit(x)** = logit function applied to x
- ❖ **sigm(x)** = sigmoid function applied to x
- ❖ **log(x)** or **ln(x)** = natural log of x
- ❖ **log1p(x)** = $\log(1 + x)$
- ❖ **expm1(x)** = $\exp(x) - 1$
- ❖ **phi(x)** = normal cumulative distribution function of x
- ❖ **iphi(x)** or **probit(x)** = inverse normal cumulative distribution function of x
- ❖ **yjt(x, lambda)** = Yeo-Johnson transformation of x with optional shape parameter
- ❖ **iyjt(x, lambda)** = inverse Yeo-Johnson transformation of x with optional shape parameter
- ❖ **mean(x)** = returns the mean of x
- ❖ **mean(x, idvar)** = returns the cluster means of x computed within the grouping variable **idvar**
- ❖ **sd(x)** = returns the standard deviation of x
- ❖ **sd(x, idvar)** = returns the cluster standard deviation of x computed within the grouping variable **idvar**
- ❖ **stand(x)** or **scale(x)** = returns x standardized as a z-score
- ❖ **stand(x, idvar)** or **scale(x, idvar)** = returns x standardized as a z-score within the grouping variable **idvar**
- ❖ **center(x)** = returns x but centered. Equivalent to $(x - \text{mean}(x))$
- ❖ **center(x, idvar)** = returns x but centered within the grouping variable **idvar**. Equivalent to $(x - \text{mean}(x, \text{idvar}))$
- ❖ **max(x)** = returns the maximum of x
- ❖ **max(x, y)** = returns the row-wise maximum between x and y
- ❖ **min(x)** = returns the minimum of x
- ❖ **min(x, y)** = returns the row-wise minimum between x and y

- ❖ `vec(x)` = creates a variable filled with the scalar `x`

The following built-in functions are only available in the **TRANSFORM** command:

- ❖ `ismissing(x)` = returns missing data indicator for `x`
- ❖ `lag1(x, time)` = returns the lag 1 of `x` based on the `time` variable
- ❖ `lag1(x, time, idvar)` = returns the lag 1 of `x` within the grouping variable `idvar`, based on the `time` variable
- ❖ `ifelse(condition, value if true, value if false)` recodes a variable `X` into a new variable with two values

2.2 Blimp Commands

DATA Command

The **DATA** command specifies the input data set, which must be saved as a .csv (comma separated values) format or a whitespace (including tab) delimited file (e.g., .dat or .txt). Blimp accepts only numeric characters for data values (e.g., a nominal variable cannot have alphanumeric labels as score values), although alphanumeric characters (e.g., NA) can be used for missing value codes. Variable names can appear in the column headers, but the **VARIABLE** command (described next) must be omitted. No file path is needed if the Blimp script (the .imp file) is located in the same directory as the data. The following code block illustrates this specification.

```
DATA: mydata.dat;
```

The **DATA** command requires a full file path to the input data set that is located in a directory other than the one that contains the Blimp script. The file path should not be enclosed in quotations. The following code block reads a data file located in a directory named “research project” located on the desktop. In line with macOS and other Unix-based systems conventions, a tilde can be used to reference the user’s home directory. The following input line reads a data file from a directory within the desktop folder.

```
DATA: ~/desktop/research project/mydata.dat;
```

VARIABLE Command

The **VARIABLES** command specifies the variable names for the data set listed on the **DATA** command. in the input file. This command should not be used if the data file has variable names as column headers. The variable list may include variables that are not used in an analysis model or imputation model. The code block below illustrates a basic specification with five variables.

```
VARIABLES: y x1 x2 x3 x4;
```

A colon can be used to specify a range of variables with the same prefix but different numeric suffixes, as follows.

```
VARIABLES: y x1:x4;
```

The colon specification also works if a group of variables has a common alphanumeric string following the numeric values (e.g., a set of variables and their recoded counterparts).

```
VARIABLES: y x1:x4 x1r:x4r;
```

ORDINAL Command

The **ORDINAL** command identifies ordinal variables that appear in a **MODEL** statement. For computational efficiency, we recommend listing binary variables on the **ORDINAL** line, but these variables could also be treated as nominal. A colon can be used to specify a range of ordinal variables, as follows.

```
ORDINAL: x1:x5;
```

By default, Blimp uses a latent response variable (i.e., probit regression) framework for ordinal variables (Albert & Chib, 1993; Carpenter & Kenward, 2013; Enders et al.,

2018; Johnson & Albert, 1999), and a logistic link is an option for binary variables (Asparouhov, T., & Muthén, B. (2021; Polson, Scott, & Windle, 2013).

NOMINAL Command

The **NOMINAL** command specifies nominal variables that appear in a **MODEL** statement. Nominal variables must be represented as a single variable with numeric codes. Blimp automatically recodes the discrete responses into a set of dummy codes (or latent response difference scores, in some cases) during estimation. By default, Blimp assigns the first (lowest) code as the reference category. To change the reference category, list the numeric code of the desired reference group in parentheses following the variable's name. To illustrate, consider two nominal variables *X* and *Z*, each with codes 1, 2, and 3. The following example assigns $X = 3$ and $Z = 1$ as the reference groups.

```
NOMINAL: x(3) z;
```

For predictors with unspecified associations, Blimp uses a latent difference score (i.e., multinomial probit regression) framework for nominal variables (Albert & Chib, 1993; Carpenter & Kenward, 2013; Enders et al., 2018; Johnson & Albert, 1999), and it uses a logistic link for multicategorical nominal variables on the left side of a tilde (Asparouhov, T., & Muthén, B. (2021; Polson, Scott, & Windle, 2013).

COUNT Command

The **COUNT** command identifies count variables that appear in a **MODEL** statement. Count dependent variables have a negative binomial regression (Asparouhov, T., & Muthén, B. (2021; Polson, Scott, & Windle, 2013). Currently, incomplete count variables can only be outcomes, not predictors.

```
COUNT: y x3;
```

CLUSTERID Command

The CLUSTERID command specifies cluster-level identifier variable(s) needed for a multilevel analysis or multilevel imputation. Two-level analyses require a single identifier for the level-2 sampling unit (cluster), and three-level analyses require level-2 and level-3 identifier variables. The order of the identifier variables does not matter, as Blimp automatically determines variable levels. To illustrate, the following code block specifies a single cluster-level identifier for a two-level analysis.

```
VARIABLES: level2id y x1 x2;  
CLUSTERID: level2id;  
MODEL: y ~ x1 x2;
```

The code block below illustrates a pair of cluster-level identifiers for a three-level analysis.

```
VARIABLES: level2id level3id y x1 x2;  
CLUSTERID: level2id level3id;  
MODEL: y ~ x1 x2;
```

Blimp currently does not allow cross-classified clustering schemes.

WEIGHT Command

The WEIGHT command identifies a variable containing sampling (i.e., inverse probability) weights. Blimp's MCMC estimation routine incorporates sampling weights for single-level and two-level models. Goldstein (2011, Section 3.4.2) describes MCMC estimation for multilevel models with sampling weights. Level-1 and level-2 sampling weights are rescaled following Goldstein (2011, Section 3.4.1). At level-1, the rescaled weights within a given cluster sum to the cluster size. This is the same as the so-called "cluster" method from Asparouhov (2006).

```
VARIABLES: v1 v2 v3 wght y x1:x5;  
WEIGHT: wght;
```

MISSING Command

The **MISSING** command is used to specify the missing value code. Missing values can be coded with a single numeric (e.g., 999) or alphanumeric value (e.g., NA). The following code block specifies a numeric value of 999 as the missing data code.

```
MISSING: 999;
```

LATENT Command

The **LATENT** command is used to define latent variables (e.g., factors in a measurement model) that will be referenced in the **MODEL** section. For example, the code block below illustrates the specification for a single latent factor with three manifest indicators.

```
LATENT: yfactor;  
MODEL:  
yfactor -> y1:y3;
```

The default scaling for latent factors is described in the **MODEL** command section. Blimp treats all latent variables as missing data to be imputed, and adding the **savelatent** keyword to the **OPTIONS** line saves the estimated latent variable scores to the imputed data.

Latent variables can be specified at any level of a multilevel model. This specification references cluster-level identifier variables from the **CLUSTERID** line. For example, the code below illustrates the specification of a level-1 latent factor with three manifest indicators measured at level-1 and a level-2 latent factor with three indicators measured at level-2. The variable

```
CLUSTERID: level2id;  
LATENT: yfactor; level2id = xfactor;  
MODEL:  
yfactor -> y1:y3;  
xfactor -> x1:x3;
```

Latent variables can also be listed on separate lines as follows.

```
LATENT:
yfactor;
level2id = xfactor;
```

RANDOMEFFECT Command

The **RANDOMEFFECT** command is used to define new latent variables that equal the random intercepts and slopes from a multilevel regression model. These latent variables are referenced in the **MODEL** section, where they can be used to predict other variables in a multilevel path or structural equation model. The **RANDOMEFFECT** command is similar to the **LATENT** command except that the random intercept and slope residuals can only function as predictors and not outcomes like a latent factor.

The specification for a random effect latent variable has four components: (a) the new latent variable's name appears on the left side of the equation, (b) the target equation's outcome variable is listed after the equals sign, (c) the random slope predictor's name from the target model appears after the vertical pipe, and (d) the cluster-level identifier variable from the **CLUSTERID** command appears at the end of the line in square brackets. The generic specification is as follows.

```
RANDOMEFFECT:
newlatent = outcome variable | random predictor [CLUSTERID var];
```

To illustrate more concretely, the code block below defines a pair of new latent variables equal to the random intercept and random slope residuals from a two-level model and uses the random effects to predict an outcome.

```
CLUSTERID: cluster;
RANDOMEFFECT:
raniceps = y | 1 [cluster];
ranslopes = y | x [cluster];
MODEL:
y ~ x | x;
```

```
z ~ raniceps ranslopes;
```

TRANSFORM Command

The TRANSFORM command creates new variables that are functions of existing variables. If imputations are requested, the new variable is saved to the output data sets. The general form of the TRANSFORM command is as follows.

TRANSFORM:

```
newvar1 = expression or function;  
newvar2 = expression or function;
```

Mathematical operator symbols are * and / for multiplication and division, + and - for addition and subtraction, and ^ to raise a variable or quantity to a power. The following examples apply these operators to create new variables from an existing variable X.

TRANSFORM:

```
newvar1 = x * 2;  
newvar2 = x / 2;  
newvar3 = x + 2;  
newvar4 = x - 2;  
newvar5 = x^2;
```

Blimp also offers three ways to recode an existing variable into a new variable. The switch function is the most convenient in many situations.

TRANSFORM:

```
newvar = switch(condition1, value if true,  
                condition2, value if true,  
                condition3, value if true,  
                ...  
                value if all conditions are false);
```

To illustrate, consider the creation of a new binary variable that equals 1 if X is less than 50 and 2 if X is greater than or equal to 50. The TRANSFORM command for recoding X into a binary variable is as follows.

TRANSFORM:

```
newvar = switch(x < 50, 1, 2);
```

As a second example, suppose X is a multicategorical variable with four groups coded 1 through 4. The TRANSFORM command below creates a new three-category variable that combines codes 3 and 4 into a single group.

TRANSFORM:

```
newvar = switch(x == 1, 1, x == 2, 2, 3);
```

The second recording strategy uses Boolean statements that evaluate to a 1 or 0. To illustrate, consider the creation of a new binary variable that equals 1 if X is less than 50 and 2 if X is greater than or equal to 50. The TRANSFORM command for recoding X into a binary variable is as follows.

TRANSFORM:

```
newvar = (x < 50)*1 + (x >= 50)*2;
```

The Boolean operator $(x < 50)$ evaluates to 1 if X is less than 50 and 0 otherwise. Multiplying by 1 establishes the score value that is assigned if the first condition is true. The Boolean operator $(x >= 50)$ similarly evaluates to a 1 or 0, and multiplying by 2 establishes the score value that is assigned if the second condition is true. Summing the two components produces the desired binary variable because only one of the two terms is non-zero.

As a second example, suppose X is a multicategorical variable with four groups coded 1 through 4. The TRANSFORM command below creates a new three-category variable that combines codes 3 and 4 into a single group.

TRANSFORM:

```
newvar = (x == 1)*1 + (x == 2)*2 + (x == 3)*3 + (x == 4)*3;
```

Following the previous example, the command consists of four Boolean operators, each of which evaluates to 1 or 0 if the condition is true or false, respectively. Each

operator is multiplied by the score value that is assigned if the condition is true. Finally, summing the result of each product recodes the variable because only one condition is non-zero.

Multiple Boolean operators can be combined into a single condition by separating two operators with **and/or**. For example, the **TRANSFORM** command below creates a new three-category variable that uses an or logical argument to recode codes 3 and 4 into a single category.

TRANSFORM:

```
newvar = (x == 1)*1 + (x == 2)*2 + ((x == 3) or (x == 4))*3;
```

The **ifelse** function is a third method for recoding variables. The general form of the command has three arguments: a condition to be evaluated, the value of the new variable if the condition is true, and the value of the new variable if the condition is false.

TRANSFORM:

```
newvar = ifelse(condition, value if true, value if false);
```

To illustrate, reconsider the creation of a new binary variable that equals 1 if X is less than 50 and 2 if X is greater than or equal to 50. The **TRANSFORM** command for recoding X into a binary variable is as follows.

TRANSFORM:

```
newvar = ifelse(x >= 50, 2, 1);
```

The condition being evaluated is whether X is greater than or equal to 50, and the new variable is assigned a value of 2 if the condition is true and 1 otherwise. Multiple **ifelse** statements can be nested within each other, but the **switch** function is generally more convenient for recoding variables into three or more score values.

Global functions are described in Section 2.1. The list below reiterates these functions.

- ❖ `abs(x)` = absolute value of `x`
- ❖ `sqrt(x)` = square root of `x`
- ❖ `exp(x)` = exponential function applied to `x`
- ❖ `logit(x)` = logit function applied to `x`
- ❖ `sigm(x)` = sigmoid function applied to `x`
- ❖ `log(x)` or `ln(x)` = natural log of `x`
- ❖ `log1p(x)` = $\log(1 + x)$
- ❖ `expm1(x)` = $\exp(x) - 1$
- ❖ `phi(x)` = normal cumulative distribution function of `x`
- ❖ `iphi(x)` or `probit(x)` = inverse normal cumulative distribution function of `x`
- ❖ `yjt(x, lambda)` = Yeo-Johnson transformation of `x` with optional shape parameter
- ❖ `iyjt(x, lambda)` = inverse Yeo-Johnson transformation of `x` with optional shape parameter
- ❖ `mean(x)` = returns the mean of `x`
- ❖ `mean(x, idvar)` = returns the cluster means of `x` computed within the grouping variable `idvar`
- ❖ `sd(x)` = returns the standard deviation of `x`
- ❖ `sd(x, idvar)` = returns the cluster standard deviation of `x` computed within the grouping variable `idvar`
- ❖ `stand(x)` or `scale(x)` = returns `x` standardized as a z-score
- ❖ `stand(x, idvar)` or `scale(x, idvar)` = returns `x` standardized as a z-score within the grouping variable `idvar`
- ❖ `center(x)` = returns `x` but centered. Equivalent to $(x - \text{mean}(x))$
- ❖ `center(x, idvar)` = returns `x` but centered within the grouping variable `idvar`. Equivalent to $(x - \text{mean}(x, idvar))$
- ❖ `max(x)` = returns the maximum of `x`
- ❖ `max(x, y)` = returns the row-wise maximum between `x` and `y`
- ❖ `min(x)` = returns the minimum of `x`
- ❖ `min(x, y)` = returns the row-wise minimum between `x` and `y`
- ❖ `vec(x)` = creates a variable filled with the scalar `x`

The following functions are specific to the **TRANSFORM** command and cannot be used elsewhere:

- ❖ `ismissing(x)` = returns missing data indicator for a variable *X*
- ❖ `missing(condition, value if true)` = returns a missing value if the condition is true and a value or variable otherwise
- ❖ `lag1(x, timescores, cluster)` = in a multilevel longitudinal structure, shifts all rows of variable *X* down by one row, as indexed by a level-1 temporal predictor `timescores` nested within a cluster-level identifier variable `cluster`
- ❖ `switch(condition1, value if true, condition2, value if true, ..., value if all conditions false)` recodes existing variables
- ❖ `ifelse(condition, value if true, value if false)` recodes a variable *X* into a new variable with two values

BYGROUP Command

The **BYGROUP** command is used to perform fully conditional specification imputation (when used in conjunction with the **FCS** command) or model estimation (when used in conjunction with the **MODEL** command) for observed subgroups in the data. For example, consider a manifest (and complete) grouping variable *G* with three categories. The following code block specifies fully condition specification imputation separately for each level of *G*.

```
BYGROUP: g;  
FCS: y x1 x2;
```

Similarly, the following code block estimates a separate multiple regression model for each subgroup of *G*.

```
BYGROUP: g;  
MODEL: y ~ x1 x2;
```

Only a single categorical variable is allowed on the **BYGROUP** command, although this limitation can be bypassed by recoding multiple categorical variables into a single variable, sample size permitting. Additionally, the **BYGROUP** variable should not appear

on the **ORDINAL**, **NOMINAL**, or **MODEL** lines. Trace plots are currently unavailable with **BYGROUP** processing. Finally, you can use this command to fit multiple-group models, but Blimp does not allow between-group constraints.

FIXED Command

The **FIXED** command identifies complete predictor variables that do not require a distribution. Incomplete variables and outcome variables (variables that appear to the left of a tilde) must be random variables with a distribution. With relatively few exceptions, we recommend listing complete variables on the fixed line, as doing so speeds computations and convergence. Fixed variables listed on the **CENTERING** line will be centered at the means of the observed data (i.e., the means will not be treated as random variables to be estimated). The following code block illustrates a multiple regression analysis with two complete fixed variables, X_1 and X_2 .

```
VARIABLES: y x1 x2 x3 x4;  
FIXED: x1 x2;  
MODEL: y ~ x1 x2 x3;
```

CENTER Command

The **CENTERING** command is used to center predictor variables in regression equations. This command affects Blimp's printed estimates but has no bearing on imputations generated by the **SAVE** command. For complete variables listed on the **FIXED** line, Blimp centers variables at arithmetic averages (grand mean or group means). For all variables assigned a distribution, the **CENTERING** command treats grand means and group means as random variables to be estimated at each MCMC iteration (Enders & Keller, 2019). Any product terms specified on the **MODEL** line automatically reflect the specified centering method.

In a single-level model, there is no need to specify the type of centering because centering at the grand means is the only option. The code block below shows a basic grand mean centering specification for a single-level multiple regression model.

```
CENTER: x1 x2;
MODEL: y ~ x1 x2 x3;
```

The equivalent specification below explicitly requests grand mean centering.

```
CENTER: grandmean = x1 x2;
MODEL: y ~ x1 x2 x3;
```

Predictor variables in a multilevel regression model can be centered at the grand means or group-level cluster means (lower-level regressors only). In this case, the type of centering must be explicitly specified. The following code block illustrates a two-level regression model with a cross-level interaction where a level-1 predictor X_1 is centered at the level-2 latent group means, and a level-2 predictor X_2 is centered at its grand mean (group mean centering is not an option for variables at the highest level).

```
CLUSTERID: level2id;
CENTER: groupmean = x1.i; grandmean = x2.j;
MODEL: y ~ x1_i x2_j x1.i*x2_j | x1_i;
```

Centering specifications can also be spread over multiple lines, as follows.

```
CLUSTERID: level2id;
CENTER:
groupmean = x1_i;
grandmean = x2_j;
MODEL: y ~ x1_i x2_j x1.i*x2_j | x1_i;
```

Importantly, group mean centering reflects deviations between the raw scores and latent group means (unless the variable is complete and listed on the **FIXED** line, in which case the group means are arithmetic averages). Further, group mean centering is always performed by subtracting the latent group means at the next level of the data hierarchy. For example, if the previous analysis was a three-level model, the centering procedure would subtract X_1 scores from the level-2 latent

group means. The group means themselves can be included in the analysis model, and these latent variables can be centered just like any other predictor.

Categorical variables can also be centered (Enders & Tofighi, 2007; Yaremych & Preacher, 2021). As mentioned elsewhere, categorical predictors (binary, ordinal, or nominal) are modeled as underlying normal latent response variables. The grand and group means are also modeled on the latent metric, and listing categorical variables on the **CENTERING** command invokes a transformation that converts the latent mean to the metric required by the analysis model (Enders & Keller, 2019). For example, centering a binary predictor converts the latent grand mean to a “manifest” mean equal to the model-implied proportion of ones in the data. Applying centering to nominal variables with three or more categories can be computationally intensive because the latent mean conversion requires Monte Carlo integration at each MCMC step.

MODEL Command

The **MODEL** command typically consists of one or more univariate regression models. Blimp's modeling framework can accommodate a wide range of analyses ranging from basic multiple regression models to complicated multilevel structural equation models with interactions involving latent variables. This section describes the command, and Chapters 4 through 7 provide numerous examples.

Regression Models

Univariate regression models are the building blocks for specifying more complex multivariate models involving networks of variables—Blimp's modeling framework simply defines any multivariate model as a collection of individual univariate regressions (see Section 1.3). A univariate regression is specified by listing an outcome variable to the left of the tilde symbol and predictors (or perhaps just an intercept) to the right of the tilde. The code block below illustrates a multiple regression analysis with three predictors.

MODEL:

```
y ~ 1 x1 x2 x3;
```

Outcome variables that appear to the left of a tilde can be latent factors or manifest variables with a variety of different metrics (normal, skewed continuous, binary, ordinal, multicategorical nominal, count). With the exception of latent outcomes where means are set equal to zero, Blimp estimates the intercept by default, and the above specification can be shortened as follows.

MODEL:

```
# unspecified predictor models  
y ~ x1 x2 x3;
```

As explained in Section 1.4, supporting regression models for incomplete predictors can be explicitly specified (i.e., they can appear as outcomes to the left of a tilde), or Blimp can create them automatically. The previous code block does not list models for the regressors, so Blimp constructs them as needed for missing data handling. The examples in Chapter 3 generally adopt this specification because it is easy to implement and accommodates normal, binary, ordinal, and multicategorical nominal variables. Leaving predictor associations unspecified also facilitates centering because grand means and latent group means (multilevel models) are iteratively estimated parameters. To reiterate, regressions among the predictors are simply a device for assigning distribution to and preserving associations among incomplete covariates. These models usually are not the substantive focus, and they need not have a logical causal construction.

Alternatively, predictor models can be invoked with a sequential specification that features a cascading pattern of univariate regressions, where the first predictor's model is empty, the second predictor is regressed on the first, the third on the first and second, and so on.

MODEL:

```
# sequentially specified predictor models
```

```
x3 ~ 1;
x2 ~ x3;
x1 ~ x2 x3;
# focal analysis model
y ~ x1 x2 x3;
```

Sequential models can be specified more succinctly by listing all predictors on the left side of the same tilde.

```
MODEL:
# sequentially specified predictor models
x1 x2 x3 ~ 1;
# focal analysis model
y ~ x1 x2 x3;
```

When using the `FIXED` command to identify complete predictor variables that do not require a distribution, those predictors should only appear on the right side of a tilde in a sequential specification.

```
FIXED: x2;
MODEL:
# sequentially specified predictor models
x1 x3 ~ x2;
# focal analysis model
y ~ x1 x2 x3;
```

A sequential specification is primarily useful for two scenarios: modeling nonlinear associations among predictors and modeling skewed or count distributions. In most other situations, unspecified and sequentially specified predictor models are equivalent. To illustrate, the code below depicts a scenario where X_2 is a quadratic function of X_3 (see the later section on interactive and polynomial effects).

```
MODEL:
x3 ~ 1;
x2 ~ x3 (x3^2);
x1 ~ x2 x3;
y ~ x1 x2 x3;
```


As a second example, the following code block assigns a Yeo-Johnson normal distribution (Yeo & Johnson, 2000) that allows X_2 's distribution to be positively or negatively skewed (see the later section on functions embedded within equations).

MODEL:

```
x3 ~ 1;
yjt(x2) ~ x3;
x1 ~ x2 x3;
y ~ x1 x2 x3;
```

Lüdtke et al. (2020b) provide recommendations for ordering variables when adopting a sequential specification (see Section 1.4).

Blimp prints a table of estimates for each outcome variable in a model (i.e., every variable to the left of a tilde). By default, the tables are printed in alphabetical order. Users can specify a custom order for tables by defining equation blocks within the **MODEL** statement. Equation blocks are defined by specifying an arbitrary name for the block (which will appear on the output) followed by a colon. For example, the code below defines two equation blocks, such that the focal regression output would be the first table of results. Within a given block, order is alphabetic.

MODEL:

```
focal.regression:
y ~ x1 x2 x3;
predictor:models:
x3 ~ 1;
yjt(x2) ~ x3;
x1 ~ x2 x3;
```

With the exception of latent dependent variables, Blimp automatically estimates each equation's intercept and residual variance. In some situations, it may be necessary to explicitly mention these parameters (e.g., when imposing a constraint or labeling a parameter). The code block below explicitly references the intercept by including a 1 on the right of the tilde (the keyword `intercept` can be used in lieu of the 1), and it uses a double-headed arrow to reference the residual variance

MODEL:

```
y ~ 1 x1 x2 x3;
```

```
y ~~ y;
```

Variances can also be specified with double tildes, as follows.

MODEL:

```
y ~ 1 x1 x2 x3;
```

```
y ~~ y;
```

Discrete Outcomes

Discrete outcomes are defined on the **ORDINAL**, **NOMINAL**, and **COUNT** lines. In general, little or no further specification is needed. For example, the following code block illustrates a probit regression for a binary outcome.

```
ORDINAL: y;
```

```
MODEL:
```

```
y ~ x1 x2 x3;
```

A logistic regression additionally applies the logit function to the dependent variable, as shown below.

```
ORDINAL: y;
```

```
MODEL:
```

```
logit(y) ~ x1 x2 x3;
```

Discrete Predictors

Discrete predictors are defined on the **ORDINAL**, **NOMINAL**, and **COUNT** lines (the latter is only available with a sequential specification). In general, little or no further specification is needed to invoke a discrete predictor. For example, the following code block illustrates a linear regression where X_2 is a binary dummy code.

```
ORDINAL: x2;
```

```
MODEL:
```

```
y ~ x1 x2 x3;
```

The discrete scores appear in the focal analysis model, but Blimp uses a latent response variable formulation for the predictor's supporting regression model (which is left unspecified above).

The specification for nominal variables is similar. To illustrate, the code block below specifies a linear regression where X_2 is a multicategorical predictor ($X_2 = 1, 2, 3$).

```
NOMINAL: x2;  
MODEL:  
y ~ x1 x2 x3;
```

Blimp uses a latent difference variable formulation (multinomial probit model) for the predictor's supporting regression (which is left unspecified above), but a set of dummy codes appear in the focal analysis model. By default, Blimp assigns the first (lowest) numeric code as the reference category. To override this default behavior, list the desired reference group's numeric code in parentheses on the **NOMINAL** line. To illustrate, the following code block assigns $X_2 = 3$ as the reference category.

```
NOMINAL: x2(3);  
MODEL:  
y ~ x1 x2 x3;
```

In some situations, it may be necessary to refer to a specific dummy code (e.g., when constraining or labeling a parameter). This specification uses a period and a numeric label following the variable's name. For example, the following code block assigns $X_2 = 3$ as the reference group, and it explicitly references the dummy codes for the $X_2 = 1$ and 2 categories in a **MODEL** statement.

```
NOMINAL: x2(3);  
MODEL:  
y ~ x1 x2.1 x2.3 x3
```

Interaction and Polynomial Terms

Traditional modeling frameworks that assume a multivariate distribution for the analysis variables (e.g., all structural equation models based on multivariate normal distribution) are fundamentally incompatible with incomplete nonlinear effects. This includes models with incomplete interaction effects, curvilinear effects, and random slopes (two- or three-level models). Practically speaking, incompatibility means that imputations generated by a multivariate distribution are mathematically impossible given the configuration of effects in the focal analysis model.

Blimp's estimation architecture avoids this problem by working with a set of univariate regression models that are guaranteed to be mutually compatible. Rather than imputing the product directly, Blimp uses a Metropolis sampling step to select imputations that are consistent with any nonlinear effects in the univariate regression models. The methodological literature uniformly favors this strategy over so-called just-another-variable imputation schemes that apply normal distribution assumptions to incomplete nonlinear effects (Bartlett et al., 2015; Enders et al., 2020; Erler et al., 2016; Kim, Belin, & Sugar, 2018; Kim, Sugar, & Belin, 2015; Lüdtke, Robitzsch, & West, 2019; Zhang & Wang, 2017). The specifications described below are the same for single-level and multilevel regression models.

Interaction terms are specified by connecting two predictors in the same equation with an asterisk. The following code block illustrates a two-way interaction with lower-order terms. The supporting regressions for incomplete predictors are constructed automatically (see Section 1.4).

MODEL :

```
y ~ x1 x2 x1*x2;
```

Similarly, the code below shows a three-way interaction with all possible two-way interactions and lower-order terms.

MODEL:

```
y ~ x1 x2 x3 x1*x2 x1*x3 x2*x3;
```

Generally speaking, any variable to the left of a tilde (dependent variables, predictors in a factored regression specification) can have interaction effects in its regression model.

Blimp allows for interactions with categorical predictors defined on the **NOMINAL** and **ORDINAL** lines. Binary and ordinal predictors function as numeric variables when multiplied by another variable; the supporting regressor model again uses a latent response variable formulation. Interactions involving multicategorical nominal variables require product terms for each dummy code in a set. By default, Blimp automatically creates a model that includes the necessary product terms. To illustrate, the code block below illustrates an interaction effect where X_2 is a multicategorical nominal predictor ($X_2 = 1, 2, 3$) and X_3 is continuous.

```
NOMINAL: x2;
```

MODEL:

```
y ~ x1 x2 x3 x2*x3;
```

In this case, Blimp automatically generates a model with two product terms, one for each of the two dummy codes (recall that $X_2 = 1$ is the reference group). In some situations, it may be necessary to refer to a specific component of the product (e.g., when constraining or labeling a parameter). The following specification is equivalent to the one above.

```
NOMINAL: x2;
```

MODEL:

```
y ~ x1 x2 x3 x2.2*x3 x2.3*x3;
```

A polynomial term in a curvilinear regression is just interaction between a variable and itself. As such, these terms can be specified by connecting a regressor with itself using an asterisk. The following code block illustrates a quadratic function with a lower-order term and a covariate.

MODEL :

```
y ~ x1 x1*x1 x2;
```

Alternatively, the quadratic term can be specified by using a function embedded in a regression equation, as follows..

MODEL :

```
y ~ x1 (x1^2) x2;
```

Correlations and Residual Correlations

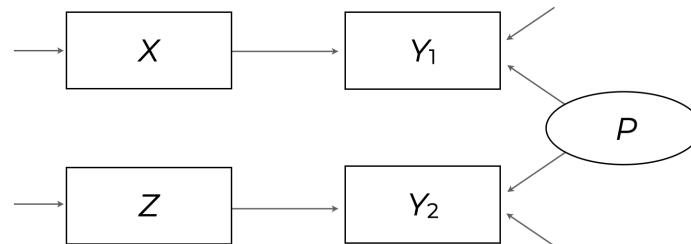
In Blimp, univariate regression models are always the building blocks for specifying more complex multivariate models involving networks of variables—the modeling framework simply defines a multivariate model as a collection of individual univariate regressions (see Section 1.3). Because Blimp does not work with the joint distribution of the variables (e.g., impose a multivariate normal distribution on the data), these univariate equations are uncorrelated by construction. For example, the code block below illustrates a bivariate analysis with two empty regression equations, but the correlation (or covariance) between the two dependent variables is not a byproduct of estimation.

MODEL :

```
y1 ~ 1;  
y2 ~ 1;
```

Blimp has two ways of estimating associations among a set of variables like those above. Whenever possible, the software assigns a multivariate distribution with a mean vector and covariance matrix as the parameters. The covariance matrix approach invokes a Wishart prior distribution. The second approach uses phantom latent factors to correlate dependent variables from different regression equations. Variances and correlations are the parameters of this specification. In a single-level model, the procedure is the “srs” specification described in Merkle and Rosseel (2018), and Blimp extends their approach to two- and three-level models. The

phantom variable approach uses a so-called separation strategy (Barnard, McCulloch, & Meng, 2000; Liu, Zhang, & Grimm, 2016) that assigns distinct priors to the diagonal and off-diagonal elements of the covariance matrix. A path diagram of the underlying model is shown below.



The multivariate structure of this specification consists of variances (or residual variances) and correlations (or residual correlations). If desired, covariances can be obtained by using the `PARAMETERS` command to define these quantities as auxiliary functions of the estimated parameters. The phantom variable approach can be manually specified by listing the `use_phantom` keyword on the `OPTIONS` line.

Like variances, correlations and residual correlations are specified with double-headed arrows or double tildes. The following code block illustrates a simple bivariate analysis with two empty regression models.

MODEL:

```

y1 ~ 1;
y2 ~ 1;
y1 ~~ y2;

```

MODEL:

```

y1 ~ 1;
y2 ~ 1;
y1 ~~ y2;

```

The analysis can be specified more succinctly as follows.

MODEL:

```

y1 ~~ y2;

```

The same specification applies to correlated residual terms from a multivariate regression.

MODEL:

```
y1 ~ x1 x2 x3;
y2 ~ x1 x2;
y1 ~~ y2;
```

Finally, multiple correlations can be specified by listing a set of variables on each side of a double-headed arrow or double tilde. The following code block requests all possible correlations among a set of five variables.

MODEL:

```
y1:y3 x1 x2 ~~ y1:y3 x1 x2;
```

Parameter Constraints

Blimp allows for many types of parameter constraints. These restrictions are imposed by listing the @ symbol and a numeric value or label following a variable's name. For example, the following code block uses a label "beta" to specify an equality constraint on X_1 and X_2 's regression slopes.

MODEL:

```
y ~ x1@beta x2@beta x3;
```

As a second example, the code below uses a numeric label to fix the regression intercept to zero during estimation.

MODEL:

```
y ~ 1@0 x1 x2;
```

Similarly, the code below fixes the variance of a variable to one during estimation.

MODEL:

```
y ~ x1 x2;
y ~~ y@1;
```


Many, but not all model parameters can be constrained. For example, between-group constraints are not permissible when using BYGROUP processing.

Auxiliary Variables

Blimp uses a sequential specification to incorporate auxiliary variables into a model. Associations among the auxiliary variables and analysis variables follow the same cascading pattern of univariate models used to connect regressors; the first auxiliary is regressed on the analysis variables, the second auxiliary variable is regressed on the first plus the analysis variables, the third is regressed on the first two, and so on. The code block below illustrates a multiple regression analysis with three auxiliary variables, A_1 to A_3 .

MODEL:

```
y ~ x1 x2;
a1 ~ y x1 x2;
a2 ~ a1 y x1 x2;
a3 ~ a1 a2 y x1 x2;
```

The auxiliary models can be specified more succinctly by listing all auxiliary variables on the left side of the same tilde.

MODEL:

```
y ~ x1 x2;
a3 a2 a1 ~ y x1 x2;
```

Latent Variables

The LATENT command described earlier defines latent variables (e.g., factors in a measurement model) referenced in the MODEL section. To illustrate, the following code block shows a basic measurement model with a single latent factor and three normally distributed indicators (indicators can also be binary or ordinal).

```
LATENT: yfactor;
MODEL:
```

```
yfactor -> y1:y3;
```

By default, Blimp establishes identification by fixing the first factor loading to one and the latent mean (or intercept) to zero. The following code block uses univariate regression equations to achieve an identical specification.

```
LATENT: yfactor;
MODEL:
yfactor ~ 1@0;
y1 ~ yfactor@1;
y2 ~ yfactor;
y3 ~ yfactor;
```

It may be beneficial to override the default identification settings in some cases. For example, convergence speed may be improved by scaling the latent factor to an indicator with complete data (or the indicator with the least amount of missing data) or fixing one of the regression intercepts instead of the latent mean to zero. To illustrate, the code block below illustrates a specification with the following features: (a) Y_1 's loading is freely estimated, (b) the latent mean is estimated, (c) Y_3 's measurement intercept is constrained to one, and (d) Y_3 's loading is constrained to one.

```
LATENT: yfactor;
MODEL:
# estimate the latent mean
yfactor ~ 1;
# estimate loadings
y1 ~ yfactor;
y2 ~ yfactor;
# fix intercept to 0 and loading to 1
y3 ~ 1@0 yfactor@1;
```

Blimp's univariate modeling framework treats latent factors as incomplete variables to be imputed (adding the `saveLatent` keyword to the `OPTIONS` line saves the estimated latent scores to the imputed data sets). Imputing the latent scores opens up interesting opportunities not available in other software packages. For example,

Blimp allows a latent variable to interact with a manifest variable or with another latent variable (Keller, 2021). The following code block illustrates a latent-by-manifest variable interaction.

```
LATENT: xfactor;  
MODEL:  
xfactor -> x1:x3;  
y ~ xfactor z xfactor*z;
```

The manifest variable Z is normal in this example, but it could have any metric that Blimp supports. Similarly, two latent variables can interact with one another. The following code block illustrates a latent-by-latent interaction involving two factors with three indicators each.

```
LATENT: xfactor mfactor;  
MODEL:  
xfactor -> x1:x3;  
zfactor -> z1:z3;  
y ~ xfactor zfactor xfactor*zfactor;
```

Finally, an outcome variable (manifest or latent) can be a polynomial function of a latent variable. The code below shows a latent variable with a quadratic effect on the outcome.

```
LATENT: xfactor;  
MODEL:  
xfactor -> x1:x3;  
y ~ xfactor (xfactor^2) z;
```

Multilevel Regression Models

Multilevel regression models require relatively few additional specifications beyond those for single-level regression models. Blimp automatically determines the level at which a variable is measured in a multilevel data set, so the user need only provide a basic model specification. The one exception is latent variables, the levels of which must be specified in the `LATENT` command. Enders et al. (2020) provide specific

details about Blimp's multilevel modeling framework, which uses the same factored regression specification outlined for single-level models. Predictor variables can be centered at their grand means or group means (Enders & Tofighi, 2007) using the **CENTERING** command (discussed later). By default, Blimp centers using latent group means (Lüdtke et al., 2008).

Section 1.5 described Blimp's treatment of incomplete predictors (exogenous variables that do not appear to the left of a tilde (in a path diagram, variables that do not have incoming arrows). When predictors are complete, there is usually no reason to specify a distribution for these variables. Instead, the covariate data essentially function as known constants, as in ordinary least squares. In contrast, incomplete predictors require an explicit distribution for imputation. Blimp allows these distributions to have many forms (e.g., normal, skewed, discrete). In most cases, assigning a distribution to a predictor means making that regressor a dependent variable in its own regression model. These supporting models can be explicitly specified, or Blimp can create them automatically. The situation with predictors is somewhat more complicated in multilevel models because lower-level regressors generally have two sources of variability. The following rules dictate Blimp's treatment of predictors in multilevel models.

- ❖ If any predictor is incomplete, then all predictors are assigned explicit supporting models
- ❖ Specifying the **SIMPLE** command for conditional effects induces an explicit model for all predictors, complete or incomplete.
- ❖ If all predictors are complete, then they are treated as fixed (no distribution or supporting model), unless group mean centering is requested on the **CENTERING** command
- ❖ Any level-1 predictor variable with group mean centering will have its latent group means estimated via an explicit between-cluster model
- ❖ Any level-1 predictor variable with its latent group means referenced using the **.mean** function will have its means estimated via an explicit between-cluster model

- ❖ Manifest (arithmetic) group means are available only for complete variables. The group means can be computed as a new variable using the **TRANSFORM** command (e.g., `xmeans = mean(x, l2id);`)
- ❖ Grand mean centering does not affect whether a predictor variable is automatically assigned a distribution
- ❖ If the ICC is exactly zero or nearly zero (e.g., a time variable in a growth model), then no between-cluster model is invoked because such a model would produce an immediate convergence failure (Blimp issues a warning message)

Blimp automatically adds random intercepts residuals to all lower-level models (outcomes or predictors) whenever the **CLUSTERID** command is used to specify a cluster-level identifier variable. To illustrate, consider a two-level regression model where X_1 and X_2 are level-1 and level-2 predictors, respectively. The following code block illustrates a regression model with random intercepts.

```
CLUSTERID: level2id;
MODEL:
y ~ x1_i x2_j;
```

The estimated model includes a random intercept in the analysis model as well as in X_1 's supporting model. In some cases, it may be necessary to manually reference the random intercept (e.g., when labeling or constraining the parameter). In the code block below, the 1 to the right of the vertical pipe represents a random intercept.

```
CLUSTERID: level2id;
MODEL:
y ~ x1_i x2_j | 1;
```

Random slope coefficients are specified by listing lower-level predictors to the right of a vertical pipe. For example, the code block below illustrates a regression model with random intercepts (implicit) and a random slope for the level-1 predictor X_1 .

```
CLUSTERID: level2id;
MODEL:
y ~ x1_i x2_j | x1_i;
```

Blimp estimates an unstructured variance–covariance matrix for the random intercepts and random slopes. Adding the `saveLatent` keyword to the `OPTIONS` line saves the random effect estimates to the imputed data sets.

Random intercepts and slopes can also appear as regressors in other equations. To illustrate, the code block below uses the `RANDOMEFFECT` command to define the intercepts and slopes as cluster-level latent variables that predict another variable Z .

```
CLUSTERID: level2id;
RANDOMEFFECT:
raniceps = y | 1 [level2id];
ranslopes = y | x1_i [level2id];
MODEL:
y ~ x1_i x2_j | x1_i;
z ~ raniceps ranslopes;
```

Blimp can also estimate three-level models. To illustrate, consider a three-level model where X_1 and X_2 are level-1 and level-2 regressors, respectively, and X_3 is a level-3 predictor. As before, Blimp automatically detects the level at which a variable is measured. The following code block illustrates a three-level regression model with random intercepts induced by a pair of cluster-level identifier variables.

```
CLUSTERID: level2id level3id;
MODEL:
y ~ x1_i x2_j x3.k;
```

As a second example, the code block below illustrates a three-level random slope model where the influence of the level-1 regressor X_1 varies across level-2 and level-3 units and the influence of the level-2 predictor X_2 varies across level-3 units.

```
CLUSTERID: level2id level3id;
MODEL:
y ~ x1_i x2_j x3.k | x1_i x2_j;
```

By default, Blimp estimates an unstructured variance-covariance matrix of the random effects at all higher levels of the data hierarchy.

In some situations, it is desirable or necessary to override Blimp's default behavior and fix certain variance components to zero (or alternatively, select which variances get estimated). This is achieved by listing the desired random effects on the right side of the vertical pipe and appending to the effect's name a cluster-level identifier in square brackets. To illustrate, the following code block illustrates a three-level model with random intercepts at both levels and a random coefficient for X_1 at the second level.

```
CLUSTERID: level2id level3id;
MODEL:
y ~ x1_i x2_j x3.k | 1[level2id] 1[level3id] x1[level2id];
```

The resulting variance–covariance matrix at level-2 is an unstructured 2×2 matrix, and the level-3 covariance matrix reduces to a scalar with only a random intercept variance. When using this specification, omitting a random effect from the right side of the vertical pipe implicitly sets its variance and covariances to zero.

Multilevel regression models can also include cluster means as group-level predictors (i.e., contextual effects; Longford, 1989; Raudenbush & Bryk, 2002). Appending the `.mean` keyword to the end of a lower-level covariate's name references that variable's latent group means. To illustrate, the following code block specifies a two-level regression model that includes X_1 as a level-1 predictor and its group means as a level-2 predictor.

```
CLUSTERID: level2id;
MODEL:
y ~ x1 x1.mean x2;
```

Importantly, the group means are cluster-level latent variables rather than deterministic arithmetic averages of the level-1 scores. Methodology research favors a latent variable specification because it can reduce bias associated with arithmetic or “manifest” group means in some scenarios (Hamaker & Muthén, 2019; Lüdtke et al., 2008).

In a three-level model, appending the `.mean` suffix to a level-1 predictor automatically introduces the level-2 and level-3 latent group means as predictors. To specify the group means at one level but not the other, additionally append the cluster-level identifier variable in square brackets. For example, the following code block illustrates a three-level random intercept regression with X_1 's level-3 latent group means as a predictor but not its level-2 averages.

```
CLUSTERID: level2id level3id;
MODEL:
y ~ x1 x1.mean[level3id] x2 x3;
```

Functions Embedded in Equations

Blimp allows users to embed functions inside parentheses on the right side of regression equations and, in limited cases, on the left side as well. As an example, the following code block features a predictor centered at a constant value of 10.

```
MODEL:
y ~ (x1 - 10);
```

The next example uses an embedded function to specify a curvilinear regression where the outcome is a quadratic function of the predictor.

```
MODEL:
y ~ x (x^2);
```

Embedded functions can also reference multiple variables. For example, the following code block defines the predictor variable as the sum of four ordinal variables. A common application occurs with scale scores computed as the sum of several questionnaire items.

```
ORDINAL: x1:x4;
MODEL:
y ~ (x1 + x2 + x3 + x4);
```


Importantly, the embedded sum function does *not* imply a deterministic computation or passive imputation. Rather, the sum is essentially a random variable that varies as a function of its four components (e.g., items). Any missing data handling uses a Metropolis sampling step to impute the individual components while simultaneously accounting for the fact that the incomplete component is part of a larger summation. The procedure for treating missing data is described in Alacam, Enders, Du, and Keller (2023).

There are two equivalent ways to specify a sum function in a regression equation. The first is to separate the first and last variables in the sum with `:+:` as shown below.

```
ORDINAL: x1:x4;
MODEL:
y ~ x1:+:x4;
```

The second method defines *the function or computation* as a new variable and then uses that new variable in a regression equation. The code block below illustrates this approach, which is equivalent to the previous two.

```
ORDINAL: x1:x4;
MODEL:
xsum = x1:+:x4;
y ~ xsum;
```

Again, it is important to emphasize that `xsum` is itself a random variable rather than a deterministic computation or passive imputation.

Although computationally different, the previous sum functions are conceptually equivalent to placing equality constraints on item-level coefficients from the same scale, as follows.

```
ORDINAL: x1:x4;
MODEL:
y ~ x1@beta x2@beta x3@beta x4@beta;
```

Embedded functions can also be part of interactive effects. To illustrate, the following code block shows an interaction between a scale (sum) score involving five items and a continuous moderating variable M (manifest or latent).

```
ORDINAL: x1:x4;
MODEL:
y ~ x1+:x4 m (( x1+:x4 ) * m);
```

Alternatively, the same analysis can be expressed by first defining the sum function as a new random variable as follows.

```
ORDINAL: x1:x4;
MODEL:
xsum = x1+:x4;
y ~ xsum m xsum*m;
```

Although computationally different, the embedded function above is conceptually equivalent to placing equality constraints on products involving items and the moderator, as follows.

```
ORDINAL: x1:x5;
MODEL:
y ~ x1@beta1 x2@beta1 x3@beta1 x4@beta1 m x1*m@beta3 x2*m@beta3
x3*m@beta3 x4*m@beta3;
```

Extending the previous idea, the code below shows the interaction between two scale scores, one computed as the sum of four ordinal items and the other computed as the sum of six items.

```
ORDINAL: x1:x4 m1:m6;
MODEL:
y ~ x1+:x4 m1+:m6 (( x1+:x4 ) * ( m1+:m6 ));
```

Alternatively, the same analysis can be expressed by first defining the sum function as a new random variable as follows.

```

ORDINAL: x1:x4;
MODEL:
xsum = x1+:x4;
msum = m1+:m6;
y ~ xsum msum xsum*msum;

```

Blimp also allows embedded functions on the left side of the tilde, but the range of allowable functions is limited (e.g., to basic mathematical operations and transformations). Moreover, the embedded function can only reference a single (dependent) variable. A common application of this functionality involves transformations to a skewed outcome variable. As an example, the following code block applies a natural log transformation to a positively-valued dependent variable.

```

MODEL:
ln(y) ~ x1 x2 x3;

```

As a second example, the code below applies the Yeo-Johnson (Yeo & Johnson, 2000) transformation to a skewed outcome variable.

```

MODEL:
yjt(y) ~ x1 x2 x3;

```

The Yeo-Johnson procedure estimates the shape of the data as the MCMC algorithm iterates and produces imputations from a skewed distribution. The analysis examples in Chapter 3 illustrate the procedure in more detail.

The description of the `TRANSFORM` command in Section 2.2 outlined Boolean operators (true/false functions) that can be used to recode an existing variable into a new variable. Boolean operators can also be used to define predictors in a regression equation. To illustrate, consider a binary dummy variable that equals 0 if X is less than 50 and 1 if X is greater than or equal to 50. In lieu of using the `TRANSFORM` command to compute a new variable, listing a Boolean operator as a predictor similarly creates a dummy variable predictor.

MODEL:

```
y ~ (x >= 50);
```

As explained previously, the operator evaluates to 1 or 0 if the condition in parentheses is true or false, respectively. Note that variables created with **TRANSFORM** are saved to the imputed data sets, whereas variables created via inline functions are not. When X has missing values, the continuous version of the variable links to other predictors, whereas the discrete version predicts Y . The imputation algorithm uses a Metropolis sampling step that allows X to simultaneously function as a continuous and binary variable.

SIMPLE Command

The **SIMPLE** command is used to request conditional effects (e.g., simple intercepts and simple slopes) from a regression model with an interaction effect. At each MCMC iteration, Blimp computes conditional effects by applying an appropriate contrast vector to the analysis model's regression coefficients. These additional auxiliary parameters thus have their own distribution, credible intervals, et cetera. The **PARAMETERS** command described next can also be used to compute contrasts.

The code block below shows the basic specification where the **SIMPLE** command requests the conditional effects of X (the focal predictor) at different values of M (the moderator).

```
CENTER: x m;
```

```
MODEL: y ~ x m x*m;
```

```
SIMPLE: x | m;
```

Multiple sets of conditional effects can be separated by semicolons

```
CENTER: x m;
```

```
MODEL: y ~ x m x*m;
```

```
SIMPLE: x | m; m | x;
```

or listed on separate lines, as follows.

```

CENTER: x m;
MODEL: y ~ x m x*m;
SIMPLE:
x | m;
m | x;

```

When a continuous moderator is listed to the right of the vertical pipe, Blimp automatically reports conditional effects at zero, plus or minus one, and plus or minus two standard deviations from zero. We highly recommend centering the focal predictor and moderator such that zero represents the mean. In a multilevel model, the standard deviation is determined by the type of centering. A continuous moderator centered at its group means has only within-cluster variation, so the pooled within-cluster standard deviation is used. A continuous moderator centered at its grand mean has both within-cluster and between-cluster variation, so the total standard deviation is used. The number of standard deviation units can also be specified. For example, the code block below requests the simple slopes of X at one half of a standard deviation above and below the mean of M .

```

CENTER: x m;
MODEL: y ~ x m x*m;
SIMPLE:
x | m@.5SD;
x | m@-.5SD;

```

When a nominal moderator variable is listed to the right of the vertical pipe, Blimp automatically computes and reports conditional effects for every group. When an ordinal variable is listed to the right of the vertical pipe, the pick-a-point score values must be specified. To illustrate, the following code block specifies conditional effects at $M = 0$ and $M = 1$ (e.g., conditional effects at each level of a binary dummy code). The same method identifies specific points for continuous moderators.

```

CENTER: x m;
MODEL: y ~ x m x*m;
SIMPLE:
x | m@0;

```

```
x | m@1;
```

The main restriction with the SIMPLE command occurs in models with multiple equations. In this case, a dependent variable in one equation can serve as the moderator in another equation, but the user must specify the values being conditioned on. The code block below shows a specification where M is the dependent variable in one equation and a moderator in the other. The variable M appears to the right of the vertical pipe along with the fixed values to condition in (i.e., default standard deviation units are not an option).

```
CENTER: x m;
MODEL:
m ~ x z;
y ~ x m x*m;
SIMPLE:
x | m@0;
x | m@1;
```

No such specification is necessary if the dependent variable in one equation is the focal predictor in the other (i.e., appears to the left of the vertical pipe). The code block below shows a specification where M is the dependent variable in one equation and the focal predictor in the other. The variable X appears to the right of the vertical pipe, and the output would return the conditional effect of M at standard deviation units above and below X 's mean.

```
CENTER: x m;
MODEL:
m ~ x z;
y ~ x m x*m;
SIMPLE:
m | x;
```

PARAMETERS Command

The PARAMETERS command is used to (a) define auxiliary parameters that are functions of a model's estimated parameters, and (b) specify custom prior

distributions. The command uses the same mathematical operators and accesses the same functions as the `TRANSFORM` command described earlier. Auxiliary parameters are computed by attaching alphanumeric labels to model parameters, then using those labels in an equation that defines a new parameter. Auxiliary parameters are computed at every MCMC iteration, and thus they have their own distributions and summary tables in the output.

As a first example, recall from the `MODEL` command section that Blimp links dependent variables from separate equations with correlations or residual correlations (instead of covariances). One use of the `PARAMETERS` command is to compute covariances. The code block below labels the variances and correlation and uses the labels to compute the covariance, which is the product of the correlation and the standard deviations.

```
MODEL:
y ~~ y@yvar;
x ~~ x@xvar;
y ~~ x@yxcorr;
PARAMETERS:
yxcov = yxcorr * sqrt(yvar * xvar);
```

As a second example, the following code block labels the three slope coefficients in a moderated regression model and uses the `PARAMETERS` command to compute the conditional effect of X (i.e., simple slope) at values of $M = 0$ and $M = 1$.

```
MODEL:
y ~ x@beta1 m@beta2 x*m@beta3;
CENTER: x m;
PARAMETERS:
m0 = 0;
m1 = 1;
slope.at.m0 = beta1 + m0 * beta3;
slope.at.m1 = beta1 + m1 * beta3;
```

As a final example, the code block below labels pathways from a single-mediator model and uses the `PARAMETERS` command to compute the product of coefficients estimator (i.e., the product of the X to M and M to Y paths; Mackinnon, 2008).

```
MODEL:
m ~ x@apath;
y ~ x@cpath m@bpath;
PARAMETERS:
indirect = apath * bpath;
```

The second major use for the `PARAMETERS` command is to introduce custom prior distributions. This functionality is currently restricted to the following list of univariate prior distributions.

- ❖ `normal(mu, var)` or `N(mu, var)` = normal distribution with μ as the mean and var as the variance
- ❖ `invgamma(a, b)` = inverse gamma distribution with a = scale (i.e., α ; prior degrees of freedom $\div 2$) and b = shape (i.e., β ; prior sums of squares $\div 2$)
- ❖ `gamma(k, theta)` = gamma distribution with k = shape and θ = scale
- ❖ `uniform(a, b)` or `unif(a, b)` = uniform distribution with lower bound a and upper bound b . Note that $-\text{Inf}$ or Inf are not permissible arguments.
- ❖ `beta(a, b)` = beta distribution with a = α and b = β
- ❖ `laplace(mu, b)` = laplace distribution with μ = location and b = scale
- ❖ `cauchy(a, g)` = cauchy distribution with a = location and g = scale
- ❖ `truncate(a, b)` or `trunc(a, b)` = truncate function to generate truncated distributions with a = lower bound and b = upper bound. To obtain one sided truncation, you can set either parameter to $-\text{Inf}$ or Inf for positive and negative infinity.

To illustrate, the following code block shows a simple regression model with informative normal priors on the regression coefficients and an inverse gamma prior for the variance with $a = 1$ and $b = .5$ (i.e., 2 additional degrees of freedom and unit sum of squares). This prior specification for the variance is identical to listing `prior1` on the `OPTIONS` line.

MODEL:

```
y ~ 1@beta0prior x@beta1prior;
```

```
y ~~ y@resvarprior;
```

PARAMETERS:

```
beta0prior ~ normal(2,20);
```

```
beta1prior ~ normal(5,10);
```

```
resvarprior ~ invgamma(1,.5);
```

In addition to the mathematical operators and functions described in the **TRANSFORM** command section, the **PARAMETERS** command can also access the following model-predicted variance expressions. These expressions could be used, for example, to create custom R^2 statistics beyond those included in the default output.

- ❖ `varname.totalvar` = model-predicted total variance of an outcome variable (a variable to the left of a tilde) named `varname`
- ❖ `varname.coefvar` = explained variance in an outcome variable named `varname` by the fixed effects coefficients
- ❖ `varname.slopevar` = explained variance in an outcome variable named `varname` by the fixed effects coefficients via random slopes in a multilevel model
- ❖ `varname.iceptvar` = explained variance in an outcome variable (a variable to the left of a tilde) named `varname` by the random intercepts
- ❖ `varname.residvar` = residual variance in an outcome variable named `varname`

WALDTEST Command

Although MCMC estimation is grounded in the Bayesian statistical paradigm, one can also view posterior medians, standard deviations, and credible intervals as surrogates for frequentist point estimates, standard errors, and confidence intervals. Levy and McNeish (2023) describe this perspective as “computational frequentism”. Essentially, the researcher wants to operate within the frequentist framework, but they use MCMC to solve a difficult estimation problem. Missing data analyses are a compelling use case for computational frequentism because optimal likelihood-based solutions are not always available or easy to use. To facilitate this perspective, the **WALDTEST** command implements custom significance tests via the Bayesian Wald test described by Asparouhov and Muthén (2021).

The Wald test statistic is a chi-square variable measuring the sum of squared, standardized differences between the point estimates (posterior means) and the null hypothesis values. The test's degrees of freedom equals the number of parameters or constraints being evaluated. The chi-square is an MCMC-generated estimate of a frequentist test statistic, and the p -value is the area above the test statistic value in a central chi-square distribution. As such, the test can be used for frequentist inference. By default, Blimp prints univariate Wald tests for all parameters except variances and variance explained test statistics.

The `WALDTEST` command provides multiple ways to specify custom hypotheses. One approach is to label parameters in the `MODEL` statement and use the `WALDTEST` command to specify the null hypothesis or condition to be evaluated. The example below illustrates a test of whether two slopes simultaneously equal 0.

```
MODEL:  
y ~ x1@b1 x2@b2 x3@b3;  
WALDTEST:  
b1 = 0;  
b2 = 0;
```

Tests of multiple parameters can also be specified using the following shortcut.

```
MODEL:  
y ~ x1@b1 x2@b2 x3@b3;  
WALDTEST:  
b1:b3 = 0;
```

More than one test can be performed by specifying multiple `WALDTEST` commands. For example, the following code block yields two tests, the first of which involves a single parameter, the second of which involves two slopes.

```
MODEL:  
y ~ x1@b1 x2@b2 x3@b3;  
WALDTEST:  
b1 = 0;  
WALDTEST:
```

```
b2:b3 = 0;
```

In `rblimp`, multiple tests are specified as elements in a list, as follows.

```
waldtest = list('b1 = 0', 'b2:b3 = 0')
```

The `WALDTEST` command can also evaluate equality and other types of constraints. For example, the following code block tests an equality constraint on two regression slopes.

```
MODEL:
y ~ x1@b1 x2@b2 x3@b3;
WALDTEST:
b1 = b2;
```

Complex hypotheses are specified by listing multiple conditions in a single `WALDTEST` command. For example, the following code block evaluates whether one slope differs from 0 and whether two slopes differ from one another.

```
MODEL:
y ~ x1@b1 x2@b2 x3@b3;
WALDTEST:
b1 = b2;
b3 = 0;
```

The second way to implement the `WALDTEST` command is similar to the `MODEL` statement—it specifies a regression model. However, the model listed on the `WALDTEST` line must be nested within the model listed on the `MODEL` statement. The first way to specify the nested model is to exclude parameters from the nested model. The code block below illustrates a comparison involving a full model with three predictors and a restricted model with only an intercept.

```
MODEL:
y ~ x1 x2 x3;
WALDTEST:
y ~ 1;
```

Alternatively, a nested model can be specified by fixing parameters to desired test values by appending an @ and a numeric label to a variable or effect. For example, the following code block illustrates an equivalent specification that fixes three slope coefficients to one.

```
MODEL:
y ~ x1 x2 x3;
WALDTEST:
y ~ x1@0 x2@0 x3@0;
```

The WALDTEST command produces the output table below.

```
MODEL FIT:

Asparouhov & Muthén Wald Tests

Test #1

Wald Statistic (Chi-Square)           133.705
Number of Parameters Tested (df)      3
Probability                           0.000
```

The WALDTEST command can also compare nested models with different variances. To illustrate, the code block below shows a two-level model with random coefficients, where the WALDTEST command is used to specify a random intercept model with two fewer parameters.

```
CLUSTERID: level2id;
MODEL:
y ~ x1 x2 x3 | x1;
WALDTEST:
y ~ x1 x2 x3 | x1@0;
```

FCS Command

The FCS command invokes a fully conditional specification multiple imputation (FCS-MI) approach similar to that described by Stef van Buuren and colleagues (van Buuren, 2007; van Buuren, Brand, Groothuis-Oudshoorn, & Rubin, 2006). This

command cannot be used in conjunction with the **MODEL** command. Rather, **FCS** deploys an MCMC algorithm that cycles through incomplete variables one at a time, imputing each variable from an additive equation that features the incomplete variable regressed on all other variables listed on the **FCS** line. This algorithm makes no distinction between outcomes and regressors in the subsequent analysis model; all entities listed on the **FCS** line are simply variables to be imputed or complete variables that contribute to imputation. The **SAVE** command outputs the filled-in data sets for reanalysis using frequentist methods (Rubin, 1987). **FCS**–MI is known to introduce bias when applied to analysis models with nonlinear terms such as interactions, polynomial effects, or random coefficients. The model-based imputation routines illustrated in Chapter 3 are far superior.

To illustrate **FCS**–MI, consider a simple scenario with one continuous variable X , one binary dummy variable D , and one 7-category ordinal variable O . The code block below shows a basic script (which could also include nominal variables).

```
DATA: data.dat;  
VARIABLES: id a1:a5 x d o z;  
ORDINAL: d o;  
MISSING: 999;  
FCS: x d o;  
NIMPS: 100;  
CHAINS: 100;  
BURN: 1000;  
ITER: 10000;  
OPTIONS: savelatent;  
SAVE: stacked = imps.dat;
```

At a minimum, the **FCS** command should include all variables and effects of interest in the analysis model(s), but the list may also include additional auxiliary variables. The commands following the **FCS** line in the script are described later in this section.

Blimp's **FCS**–MI routine primarily differs from the classic MICE (Multiple Imputation by Chained Equations; van Buuren & Groothuis-Oudshoorn, 2011) approach in two

ways. First, Blimp's algorithm is a true Gibbs sampler; this is a small technical nuance that makes no difference in practice. Second, Blimp adopts a fully latent specification for all categorical variables. As noted previously, Blimp uses a probit regression framework that views discrete scores as arising from one or more normally distributed latent response variables (or latent response difference scores in the case of multicategorical nominal variables). Applied to the previous example, the binary dummy variable D and the 7-category ordinal variable O have corresponding latent response variables D^* and O^* , respectively. Blimp's FCS-MI routine uses the latent variables both as predictors and as outcomes. The round robin imputation models for this example are as follows.

$$X = \mu_X + \gamma_{11}(D^* - \mu_{D^*}) + \gamma_{21}(O^* - \mu_{O^*}) + r_1$$

$$D^* = \mu_{D^*} + \gamma_{12}(O^* - \mu_{O^*}) + \gamma_{22}(X - \mu_X) + r_2$$

$$O^* = \mu_{O^*} + \gamma_{13}(X - \mu_X) + \gamma_{23}(D^* - \mu_{D^*}) + r_3$$

The latent response models also incorporate threshold parameters that divide the latent distributions into discrete segments, and the residual variances of r_2 and r_3 are fixed at one to establish the latent variable metrics.

Listing the `savelatent` keyword on the `OPTIONS` line saves both the discrete and latent response variables to the imputed data files (by default, only the discrete imputes are written to the imputed data files). The imputed latent scores (plausible values) could be used in lieu of the discrete scores in a subsequent analysis. For example, the analysis in Section 5.7 illustrates an item-level factor analysis that uses imputed latent response scores. In a similar vein, Muthén and Asparouhov (2016) describe an application that replaces a binary mediator with a latent response variable. As an aside, the `savelatent` keyword can also be used in conjunction with imputations generated by the `MODEL` command.

If desired, listing the `mice` and `manifest` keywords on the `OPTIONS` line alters Blimp's default behaviors and invokes an algorithm that is equivalent to the one in

the MICE package in R (van Buuren & Groothuis-Oudshoorn, 2011). In addition to a slight algorithmic modification, this specification uses discrete variables as predictors on the right side of equations. The round robin imputation models for this example are as follows.

$$X = \gamma_{01} + \gamma_{11}D + \gamma_{21}O + r_1$$

$$D^* = \gamma_{02} + \gamma_{12}O + \gamma_{22}X + r_2$$

$$O^* = \gamma_{03} + \gamma_{13}X + \gamma_{23}D + r_3$$

The MICE package deploys logistic rather than probit models for categorical variables, but this distinction tends to make little difference in practice.

The multilevel version of fully conditional specification (Enders et al., 2018) automatically introduces the latent group means of all lower-level variables in the imputation model (i.e., latent contextual effects); this is true for both continuous and latent response variables. Including the group means in the imputation model allows all between-cluster associations to vary independently of the within-cluster associations. Listing the `noclmean` keyword on the `OPTIONS` line removes the latent group means from the regression models, producing a more restrictive imputation model where the within- and between-cluster regressions are assumed to be equal. For two-level models, a heterogeneous level-1 variance structure is invoked by listing the `hev` keyword on the `OPTIONS` line. This method is described in Kasim and Raudenbush (1998).

BURN Command

The `BURN` command specifies the number of burn-in iterations. Bayesian analysis results summarize estimates taken from iterations following the burn-in period, and multiple imputations (via `FCS` or `MODEL`) are saved after the burn-in period. To illustrate, the following code block illustrates a 5,000-iteration burn-in period.

```
BURN: 5000;
```

The number of burn-in iterations should always be determined by examining the potential scale reduction factor diagnostic (Gelman & Rubin, 1992) from the Blimp output. Material at the beginning of Chapter 3 describes how to use these diagnostics.

ITER Command

The ITER (also ITERATIONS) command specifies the number of iterations after the burn-in period. The tabular summaries reflect Bayesian analysis results taken from the post burn-in period. To illustrate, the following code block specifies 10,000 MCMC iterations following an initial burn-in period of 5,000 iterations.

```
BURN: 5000;  
ITER: 10000;
```

Note that the total number of iterations is distributed equally across the number of MCMC chains, the default value of which is two (see the CHAINS command). In our experience, 10,000 iterations is usually more than sufficient, but material at the beginning of Chapter 3 describes how to verify that this is the case.

CHAINS Command

The CHAINS command is used to specify the number of MCMC processes (and optionally, the number of processors used for computation). The default number of chains is two, and the total number of computational cycles specified on the ITER line is always divided equally across chains. By default, Blimp attempts to distribute MCMC chains across physical cores, resulting in faster computation (e.g., on a 10-core machine, specifying 10 chains would automatically assign one MCMC process per core). Because Blimp automatically uses the maximum available cores, this specification would primarily be used to specify fewer resources. For example, the

code block below specifies 10,000 iterations spread across 10 unique MCMC chains. The MCMC processes are completed sequentially using two physical cores.

```
ITER: 10000;  
CHAINS: 10 processors 2;
```

By default, each chain will have a different seeding value and different random starting values. Random starting values can be disabled by specifying the `norandomstarts` keyword on the `OPTIONS` line.

NIMPS Command

The `NIMPS` command is used to specify the desired number of multiple imputation data sets to save during MCMC estimation (saving imputed data sets is optional). Graham, Olchowski, and Gilreath (2007) suggest using at least 20 imputed data sets to maximize power, and other studies have shown that 100 or more imputations may be necessary to reduce the impact of Monte Carlo simulation error on standard errors and get precise estimates of confidence interval half-widths and probability values (Bodner, 2008; Harel, 2007; von Hippel, 2018). Imputations can be saved at regular intervals during a single MCMC chain, at the final iteration of multiple MCMC processes, or some combination of the two. The code block below saves 100 imputed data sets from the final iteration of 100 MCMC chains, each with 5,000 burn-in iterations and 100 iterations thereafter (i.e., 10,000 total iterations spread across 100 MCMC processes).

```
BURN: 5000;  
ITER: 10000;  
NIMPS: 100;  
CHAINS: 100;
```

THIN Command

The `THIN` command is used to specify the between-imputation interval when saving multiple imputations from the same MCMC chain. For example, the following code

block deploys two MCMC chains (the default) that create 100 filled-in data sets by saving imputations every 1,000 iterations after the 5,000-iteration burn-in period.

```
NIMPS: 100;  
BURN: 5000;  
THIN: 1000;
```

Saving multiple imputations is optional, and this command is not necessary; however, either **THIN** or **ITER** must be specified when saving filled-in data sets. The **THIN** command has no impact on printed parameter summaries, which are always based on the post burn-in iterations.

OPTIONS Command

The following keywords are used in conjunction with either the **FCS** or **MODEL** commands. Bolded keywords are default and do not require explicit specification.

- ❖ **prior1/prior2/prior** = Three common prior distributions for the residual variances and covariances of dependent variables; **prior1** is more informative because it adds to the degrees of freedom and sums of squares, **prior2** is less informative because subtracts from the degrees of freedom, and **prior3** has zero degrees of freedom and adds zero to the sums of squares
- ❖ **xprior1/xprior2/xprior3** = Three common prior distributions for the residual variances of predictor variables with unspecified associations
- ❖ **psr/nopsr** = Compute the potential scale reduction factor diagnostic
- ❖ **hov/hev** = homogenous versus heterogeneous within-cluster variances
- ❖ **randomstarts/norandomstarts** = Enable/disable random starting values for different MCMC chains
- ❖ **listwise** = Enable listwise deletion (off by default).
- ❖ **saveVariableNames** or **saveVarNames** = write variable names as column headers when saving imputed data sets.

The following keywords are used in conjunction with the **FCS** command to alter the behavior of the fully conditional specification imputation algorithm.

- ❖ **mice** = classic mice algorithm instead of a Gibbs sampler

- ❖ **manifest** = manifest categorical rather than latent response variables as predictors in the imputation model
- ❖ **noclmean** = exclude latent cluster means from level-2 (and level-3) imputation models

The following keywords are used in conjunction with the **SAVE** command to alter the composition of imputed data files.

- ❖ **savelatent** = save factor or latent variable scores (measurement models), random effects (two-level models), latent response variables (categorical variables), and normalized values from the Yeo-Johnson transformation
- ❖ **savepredicted** = save the predicted values of continuous outcomes, predicted probabilities for binary and nominal outcomes, and predicted latent response variable scores for ordinal outcomes
- ❖ **saveresidual** = save residuals (or within cluster residuals)
- ❖ **csv** = save data sets as comma separated .csv files (instead of space delimited .dat files) and write variable names as column headers

OUTPUT Command

The **OUTPUT** command is used to customize the printed parameter summaries. By default, Blimp prints the posterior median, posterior standard deviation, 95% credible interval limits, split chain potential scale reduction factor, and effective number of MCMC samples (estimated number of independent MCMC iterations using split chain approach) for each parameter. Listing any of the following keywords on the **OUTPUT** command overrides Blimp's default output tables with new tables containing the requested quantities. Although not printed by default, Wald tests for individual parameters can be requested by listing the **wald** and **pvalue** options on the **OUTPUT** command.

- ❖ **default** = posterior median, standard deviation, 95% credible interval, split chain potential scale reduction factor, effective number of MCMC samples
- ❖ **default_mean** = posterior mean, standard deviation, 95% credible interval, split chain potential scale reduction factor, effective number of MCMC samples

- ❖ `default_median` = posterior median, posterior median absolute deviation (scaled to be same metric as std. dev.), 95% credible interval, split chain potential scale reduction factor, effective number of MCMC samples
- ❖ `mean` = posterior mean
- ❖ `median` = posterior median
- ❖ `stddev` = posterior standard deviation
- ❖ `mad_sd` = posterior median absolute deviation (scaled to be same metric as the standard deviation)
- ❖ `quant` = 2.5%, 25%, 50%, 75%, 97.5% quantiles
- ❖ `quant50` = 25% and 50% quantiles
- ❖ `quant95` = 2.5% and 97.5% quantiles
- ❖ `psr` = potential scale reduction factor computed after the burn-in period
- ❖ `n_eff` = print effective number of MCMC samples
- ❖ `mcmc_se` = print MCMC simulation standard error
- ❖ `wald` = print Bayesian Wald chi-square statistics for each parameter
- ❖ `pvalue` = print p-values for Bayesian Wald chi-square tests

To illustrate, the code block below creates a custom table displaying only the median, a set of quantiles (2.5%, 25%, 50%, 75%, and 97.5%), and potential scale reduction factors computed following the burn-in period.

```
OUTPUT: median quant psr;
```

The code block below specifies Blimp's default output with the additional quantities.

```
OUTPUT: default median quant psr;
```

The final code block below specifies Blimp's default output with parameter-specific Bayesian Wald chi-square statistics and p-values.

```
OUTPUT: default wald pvalue;
```

SAVE Command

The **SAVE** command is used to save byproducts of MCMC estimation. The principal use for this command is to save multiply imputed data sets, but the command also saves parameter estimates from the burn-in and post burn-in iterations, posterior summaries, and potential scale reduction factors. Unless a full file path is specified, Blimp saves the specified files to the directory that contains the input script.

Multiple imputations can be saved in three different formats: (a) as separate data files (ideal for analysis in *Mplus* or HLM), (b) in a single stacked file with an additional identifier variable that indexes imputations (ideal for analysis in R, SPSS, and SAS), and (c) a single stacked file that includes the original data indexed with a zero value (ideal for analysis in Stata). The following code block illustrates all three specifications.

SAVE:

```
separate = imp*.dat;  
stacked =imps.dat;  
stacked0 =imps0.dat;
```

When saving imputations to separate files, the asterisk in the file path is replaced with an integer in the file name (e.g., specifying `imp*.dat` produces imputed data sets named `imp1.dat`, `imp2.dat`, `imp3.dat`, et cetera). The separate-file specification also generates a text file that contains the names of the individual data files (this file functions as the input data when analyzing imputations in *Mplus*).

The imputed data sets include all variables from the input data (regardless of whether they were used in an analysis or imputation routine) along with the values of any latent variables, predicted scores, and residuals specified on the **OPTIONS** line (the `savelatent`, `savepredicted`, and `saveresidual` keywords). The stacked format adds a variable to the first column of the data that indexes the data sets. The order of the variables in the imputed data sets is listed at the bottom of the Blimp output. The output excerpt below provides an illustration.

VARIABLE ORDER IN IMPUTED DATA:

```
stacked = 'imps.dat'

imp# id n1 d1 o1 y x1 d2 x2 x3
```

In addition to creating imputed data sets, the `SAVE` command can produce files containing the estimated parameters for burn-in iterations (`burn = filename;`), estimated parameters for the post burn-in iterations (`iterations = filename;`), posterior summaries of the parameter estimates as they appear on the Blimp output (`estimates = filename;`), starting values (`starts = filename;`), the potential scale reduction factor values for all parameters (`psr = filename;`), the Bayesian Wald test statistic (`waldtest = filename;`), and the average imputation across the post burn-in iterations (`avgimp = averageimps.dat;`). The code block below illustrates these options.

SAVE:

```
burn = burnin.dat;
iterations = iterations.dat;
estimates = estimates.dat;
starts = starts.dat;
psr = psr.dat;
waldtest = wald.dat;
avgimp = averageimps.dat;
```

When using multiple MCMC chains, chain-specific quantities can be saved by specifying an asterisk in the filename. Blimp replaces this symbol in the filename with a numeric value that indexes the chains. The following code block illustrates this specification.

SAVE:

```
burn = burnin*.dat;
iterations = iterations*.dat;
estimates = estimates.dat;
starts = starts.dat;
psr = psr.dat;
avgimp = averageimps.dat;
```

Parameter summaries and starting values are saved in a single file regardless of the number of MCMC chains used for computations.

3 Diagnosing Convergence and Specifying the Number of Iterations

Diagnosing the MCMC algorithm's convergence and determining the total number of computational cycles is an important part of any analysis. The initial burn-in (trial) period should be long enough for the algorithm to achieve independence from its random starting values and achieve a steady state (i.e., converge in distribution); the total number of iterations after the burn-in period should be large enough to provide adequate precision. This section describes this process of determining these two quantities. These steps are applicable to any analysis, including all the ensuing examples. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex3a.imp](#) [Ex3b.imp](#) [data8.dat](#)

The first step in an analysis is to perform a preliminary diagnostic run to determine the length of the burn-in period. This initial period should be long enough for the MCMC algorithm to converge. As a starting point, we find it useful to specify 10,000 burn-in cycles for the preliminary analysis. The code block below estimates a two-level random coefficient model (see Example 6.3) with this setting on the **BURN** line. The default number of chains is two, and the number of iterations after the burn-in period (the **ITER** line) is not important at this point.

```
DATA: data8.dat;  
VARIABLES: level1id level2id x1_i x2_i y_i x3_i x4_i d1_j  
              n1_j x5_j x6_j x7_j x8_j x9_j;  
CLUSTERID: level2id;  
ORDINAL: d1_j;  
MISSING: 999;  
FIXED: d1_j;  
CENTER: groupmean = x1_i; grandmean = x2_i x7_j d1_j;  
MODEL: y_i ~ x1_i x2_i x7_j d1_j | x1_i;  
SEED: 90291;  
BURN: 10000;  
ITER: 10000;
```


OPTIONS: labels;

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd1.j',
  fixed = 'd1.j',
  center = 'groupmean = x1.i;
grandmean = x2.i x7.j d1.j',
  model = ' y.i ~ x1.i x2.i x7.j d1.j | x1.i',
  seed = 90291,
  burn = 10000,
  iter = 10000,
  options = 'labels'
)
output(mymodel)
```

Blimp divides the burn-in period into 20 equal segments and computes the split-chain potential scale reduction factor (Gelman et al., 2014) at the end of each interval. The table below shows the highest (worst) potential scale reduction factor across all model parameters.

BURN-IN POTENTIAL SCALE REDUCTION (PSR) OUTPUT

NOTE: Split chain PSR is being used. This splits each chain's iterations to create twice as many chains.

Comparing iterations across 2 chains	Highest PSR	Parameter #
251 to 500	1.461	12
501 to 1000	1.303	13
751 to 1500	1.157	13
1001 to 2000	1.313	5
1251 to 2500	1.085	13
1501 to 3000	1.055	5
1751 to 3500	1.096	13
2001 to 4000	1.090	13
2251 to 4500	1.051	13
2501 to 5000	1.024	13

2751 to 5500	1.020	13
3001 to 6000	1.015	13
3251 to 6500	1.041	5
3501 to 7000	1.011	5
3751 to 7500	1.015	8
4001 to 8000	1.009	3
4251 to 8500	1.030	8
4501 to 9000	1.028	14
4751 to 9500	1.032	8
5001 to 10000	1.024	8

The table shows that the index drops to acceptable levels (e.g., less than 1.05, where 1 is the theoretical minimum) by iteration 5,000. A good rule of thumb is to set the burn-in period for the final run to a value at least as large as 5,000. If the value in the bottom row of the table (the final checkpoint) exceeds 1.05, increase the number of burn-in iterations (e.g., to 20,000) and rerun the model.

The potential scale reduction factor table indicates that the highest (worst) values prior to convergence are primarily associated with parameter numbers 13 and 5. Listing the optional `labels` keyword on the `OPTIONS` line prints a table of potential scale reduction factors for all model parameters along with their numeric indices. In some cases (e.g., latent variable models), very high potential scale reduction factors will be associated with standardized regression weights (e.g., due to scaling constraints). In general, these can be ignored, and the focus should be on the unstandardized parameters. The table for the focal regression model is shown below (unspecified predictor models also have similar tables). The table indicates that parameter numbers 13 and 5 correspond to the standardized coefficient for a level-2 predictor and the intercept, respectively. The columns of the table give the potential scale reduction factors for the final five checkpoints during the burn-in period.

PARAMETER LABELS:

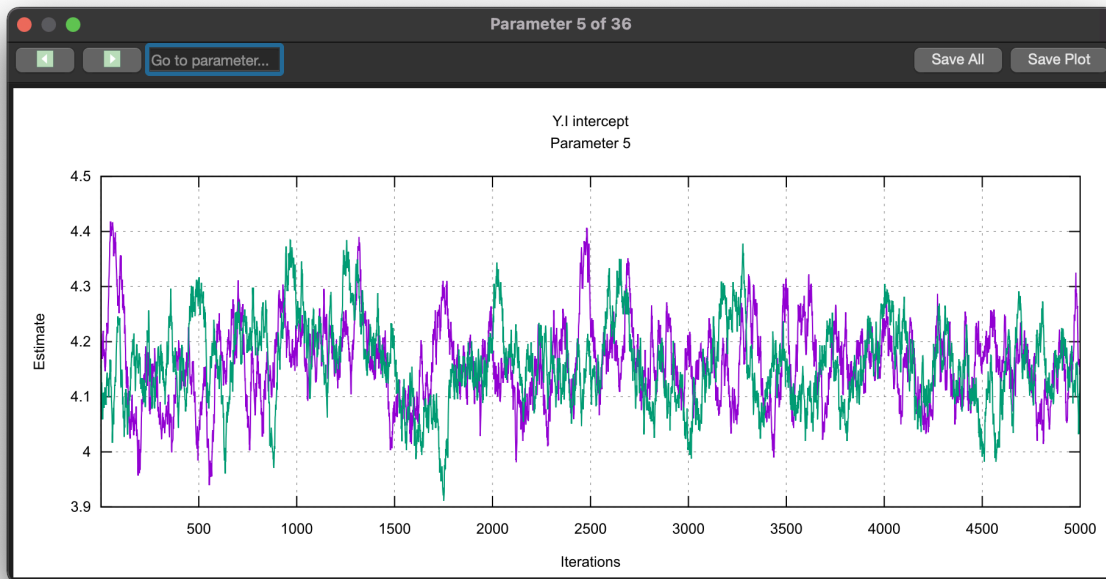
Printing out PSR for last 5 comparisons:

NOTE: Split chain PSR is being used. This splits each chain's iterations to create twice as many chains.

Comparing iterations across 2 chains
 [1] 4001 to 8000

	[2]	4251 to 8500				
	[3]	4501 to 9000				
	[4]	4751 to 9500				
	[5]	5001 to 10000				
Outcome Variable: y_i	[1]	[2]	[3]	[4]	[5]	
Variances:						
1 L2 : Var(Intercept)	1.00	1.00	1.00	1.00	1.00	1.00
2 L2 : Cov(x1.i,Intercept)	1.00	1.00	1.00	1.00	1.00	1.00
3 L2 : Var(x1.i)	1.01	1.01	1.02	1.01	1.00	1.00
4 Residual Var.	1.00	1.00	1.00	1.00	1.00	1.00
Coefficients:						
5 Intercept	1.01	1.00	1.01	1.01	1.01	1.01
6 x1_i	1.00	1.00	1.00	1.00	1.00	1.00
7 x2_i	1.00	1.00	1.00	1.00	1.00	1.00
8 x7_j	1.01	1.03	1.03	1.03	1.02	1.02
9 d1_j	1.00	1.00	1.00	1.00	1.00	1.00
Standardized Coefficients:						
10 x1_i	1.00	1.00	1.00	1.00	1.00	1.00
11 x2_i	1.00	1.00	1.00	1.00	1.00	1.00
12 x7_j	1.01	1.03	1.03	1.03	1.02	1.02
13 d1_j	1.00	1.00	1.00	1.00	1.00	1.00
Proportion Variance Explained						
14 by Coefficients	1.01	1.02	1.03	1.02	1.01	1.01
15 by Level-2 Random Intercepts	1.00	1.00	1.00	1.00	1.00	1.00
16 by Level-2 Random Slopes	1.01	1.01	1.02	1.01	1.00	1.00
17 by Level-1 Residual Variation	1.00	1.00	1.00	1.00	1.00	1.00

A trace plot of the intercept estimates from the first 5,000 computational cycles is shown below. Plot features such as the number of chains or iterations printed can be set in the Blimp Studio > Preferences pull-down menu. Plotting can also be turned off completely in these settings (this can reduce post-processing time considerably).



The next step is to set the burn-in period and total number of iterations for the final analysis. We find it useful to specify 10,000 iterations following the initial burn-in period, which for this example we set at 5,000 based on the preliminary diagnostic run. The code block below reflects these settings on the `BURN` and `ITER` line. The `labels` keyword and `OPTIONS` line are no longer needed.

```
DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i x3_i x4_i d1.j
              n1_j x5_j x6_j x7_j x8_j x9_j;
CLUSTERID: level2id;
ORDINAL: d1_j;
MISSING: 999;
FIXED: d1_j;
CENTER: groupmean = x1_i; grandmean = x2_i x7_j d1_j;
MODEL: y_i ~ x1_i x2_i x7_j d1_j | x1_i;
SEED: 90291;
BURN: 5000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd1.j',
  fixed = 'd1.j',
  center = 'groupmean = x1.i;
  grandmean = x2.i x7.j d1.j',
  model = ' y.i ~ x1.i x2.i x7.j d1.j | x1.i',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```

The Blimp output tables include point estimates and measures of uncertainty (posterior median and standard deviation), 95% credible interval limits, potential scale reduction factors for the iterations following the burn-in period, and the effective number of MCMC samples. The output tables generally include a section for variances, coefficients, standardized estimates, and variance explained effect sizes (Rights & Sterba, 2019).

OUTCOME MODEL ESTIMATES:

Summaries based on 10000 iterations using 2 chains.

Outcome Variable: y_i

Grand Mean Centered: $d1.j$ $x2.i$ $x7.j$

Group Mean Centered: $x1.i$

Parameters	Median	StdDev	2.5%	97.5%	PSR	N_Eff

Variances:						
L2 : Var(Intercept)	0.619	0.083	0.484	0.811	1.001	4793.680
L2 : Cov($x1.i$,Intercept)	0.013	0.016	-0.018	0.045	1.000	1569.242
L2 : Var($x1.i$)	0.020	0.006	0.011	0.034	1.002	715.523
Residual Var.	0.358	0.011	0.336	0.382	1.000	4620.046
Coefficients:						
Intercept	4.168	0.070	4.030	4.305	1.015	169.900
$x1.i$	-0.094	0.019	-0.132	-0.056	1.000	1841.459

x2_i	0.086	0.008	0.071	0.102	1.000	2622.345
x7_j	0.056	0.066	-0.072	0.194	1.025	149.321
d1_j	-0.104	0.150	-0.398	0.193	1.003	143.175
Standardized Coefficients:						
x1_i	-0.094	0.020	-0.133	-0.056	1.001	1870.786
x2_i	0.184	0.019	0.148	0.222	1.001	2510.830
x7_j	0.054	0.064	-0.069	0.183	1.025	151.449
d1_j	-0.050	0.071	-0.188	0.093	1.003	143.392
Proportion Variance Explained						
by Coefficients	0.061	0.017	0.038	0.105	1.011	186.000
by Level-2 Random Intercepts	0.580	0.034	0.515	0.646	1.001	2791.706
by Level-2 Random Slopes	0.020	0.006	0.011	0.034	1.003	655.997
by Level-1 Residual Variation	0.335	0.027	0.281	0.389	1.002	1828.375

The rightmost column of the table—the effective number of MCMC samples—is essentially the number of independent estimates on which the parameter summaries are based after removing autocorrelations from the MCMC process. Gelman et al. (2014, p. 287) recommend values greater than 100. All values in the example table exceed this recommended minimum. Increasing the total number of iterations would provide more precise summaries.

4 Regression Model Analysis Examples

The analysis examples in this chapter primarily illustrate different types of univariate regression models. Univariate regressions are the basic building blocks of more complicated multivariate and latent variable models, which are just collections of univariate equations. In general, it is possible to mix and match features from any examples to easily create complex analysis models that honor features of the data. The examples use a generic notation system where variable names usually consist of an alphanumeric prefix and a numeric suffix (e.g., Y , X_1 , X_{1,N_1} , D_1 , D_2). The letter Y designates a dependent variable, a D prefix denotes a binary dummy variable, an O prefix indicates an ordinal variable, and an N prefix indicates a multicategorical nominal variable. Finally, the model equations use a "cgm" superscript to indicate grand mean centering. The following list outlines the examples in this section. The following list outlines the examples in this section.

- ❖ 4.1: Correlations and Descriptive Statistics
- ❖ 4.2: Polychoric Correlations With Latent Response Variables
- ❖ 4.3: Linear Regression
- ❖ 4.4: Model-Based Multiple Imputation
- ❖ 4.5: Linear Regression With Nominal Predictors
- ❖ 4.6: Fully Conditional Specification Multiple Imputation
- ❖ 4.7: Auxiliary Variables
- ❖ 4.8: Moderated Regression With an Interaction
- ❖ 4.9: Multiple Imputation Within Subgroups
- ❖ 4.10: Curvilinear Regression
- ❖ 4.11: Probit Regression With a Binary Outcome
- ❖ 4.12: Probit Regression With an Ordinal Outcome
- ❖ 4.13: Logistic Regression With a Binary Outcome
- ❖ 4.14: Logistic Regression With a Multicategorical Outcome
- ❖ 4.15: Count Regression
- ❖ 4.16: Zero-Inflated Count Outcome

- ❖ 4.17: Scale Scores With Incomplete Item Responses
- ❖ 4.18: Scale Score Interactions
- ❖ 4.19: Skewed Predictor With a Yeo-Johnson Transform
- ❖ 4.20: Skewed Outcome With a Yeo-Johnson Transform
- ❖ 4.21: Propensity Score Estimation With Missing Data
- ❖ 4.22: Sampling Weights
- ❖ 4.23: Wald Significance Tests

4.1: Correlations and Descriptive Statistics

This example illustrates correlations and descriptive statistics. Blimp has two ways of estimating associations among a set of variables. Whenever possible, the software assigns a multivariate distribution with a mean vector and covariance matrix as the parameters. The covariance matrix approach invokes a Wishart prior distribution. The second approach uses phantom latent factors to correlate dependent variables from different regression equations. Variances and correlations are the parameters of this specification. In a single-level model, the procedure is the “srs” specification described in Merkle and Rosseel (2018), and Blimp extends their approach to two- and three-level models. A path diagram of the underlying model is shown in the **MODEL** section of Chapter 2. The phantom variable approach uses a so-called separation strategy (Barnard, McCulloch, & Meng, 2000; Liu, Zhang, & Grimm, 2016) that assigns distinct priors to the diagonal and off-diagonal elements of the covariance matrix. Computer simulation studies suggest that the separation strategy gives more accurate estimates of the variance components, although the correlation estimate may be attenuated when the number of level-2 units is small (Keller & Enders, 2021).

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.1a.imp](#) [Ex4.1b.imp](#) [Ex4.1c.imp](#) [data1.dat](#)

The following code block uses the default approach to estimate means, covariances, and correlations.

```
DATA: data1.dat;  
VARIABLES: id n1 d1 y1 y2 x1 d2 x2 x3;  
MISSING: 999;  
MODEL:  
x1 y1 y2 ~~ x1 y1 y2;  
SEED: 90291;  
BURN: 10000;  
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'data1.rda')  
  
mymodel <- rblimp(  
  data = data1,  
  model = 'x1 y1 y2 ~~ x1 y1 y2',  
  seed = 90291,  
  burn = 10000,  
  iter = 10000  
)  
output(mymodel)
```

Adding the `use_phantom` keyword to the `OPTIONS` line invokes the phantom variable approach with separation priors.

```
DATA: data1.dat;  
VARIABLES: id n1 d1 y1 y2 x1 d2 x2 x3;  
MISSING: 999;  
MODEL:  
x1 y1 y2 ~~ x1 y1 y2;  
SEED: 90291;  
BURN: 10000;  
ITER: 10000;  
OPTIONS: use_phantom;
```

Finally, the code block below labels variance parameters and uses the `PARAMETERS` command to compute standard deviations. Omitting the `use_phantom` argument defaults to a standard multivariate model.

```
DATA: data1.dat;
VARIABLES: id n1 d1 y1 y2 x1 d2 x2 x3;
MISSING: 999;
MODEL:
x1 y1 y2 ~~ x1 y1 y2;
x1 ~~ x1@varx1;
y1 ~~ y1@vary1;
y2 ~~ y2@vary2;
PARAMETERS:
sd_x1 = sqrt(varx1);
sd_y1 = sqrt(vary1);
sd_y2 = sqrt(vary2);
SEED: 90291;
BURN: 10000;
ITER: 10000;
```

4.2: Polychoric Correlations With Latent Response Variables

This example illustrates polychoric correlations among continuous variables and latent response scores from binary and ordinal variables. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.2.imp](#) [data1.dat](#)

The syntax highlights are as follows.

- ❖ `ORDINAL` command identifies binary and ordinal variables

```
DATA: data1.dat;
VARIABLES: id n1 d1 o1 y1 x1 d2 x2 x3;
ORDINAL: d1 o1;
MISSING: 999;
MODEL:
d1 o1 y1 x1 ~~ d1 o1 y1 x1;
```

```
SEED: 90291;
BURN: 25000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  ordinal = 'd1 o1',
  model = 'd1 o1 y1 x1 <-> d1 o1 y1 x1',
  seed = 90291,
  burn = 30000,
  iter = 10000
)
output(mymodel)
```

4.3: Linear Regression

This example illustrates a linear regression analysis. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.3.imp](#) [data1.dat](#)

The model features a pair of continuous predictors and a binary dummy code, as follows. The *cgm* superscript denotes variables centered at their grand means.

$$Y = \beta_0 + \beta_1 X_1^{cgm} + \beta_2 X_2^{cgm} + \beta_3 D + \varepsilon$$

The syntax highlights are as follows.

- ❖ `ORDINAL` command identifies a binary predictor
- ❖ `FIXED` command identifies a complete predictor
- ❖ `CENTER` command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables

```
DATA: data1.dat;  
VARIABLES: id v1 v2 v3 y x1 d x2 v4;  
ORDINAL: d;  
MISSING: 999;  
FIXED: d;  
CENTER: x1 x2;  
MODEL: y ~ x1 x2 d;  
SEED: 90291;  
BURN: 1000;  
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'data1.rda')  
  
mymodel <- rblimp(  
  data = data1,  
  ordinal = 'd',  
  fixed = 'd',  
  center = 'x1 x2',  
  model = 'y ~ x1 x2 d',  
  seed = 90291,  
  burn = 1000,  
  iter = 10000  
)  
output(mymodel)
```

4.4: Model-Based Multiple Imputation

Blimp can save multiple imputations from any model it estimates. This example illustrates a model-based multiple imputation procedure tailored around the linear regression model from Example 4.3. Clicking the links below downloads the scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.4.imp](#) [Ex4.4.R](#) [data1.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies a binary predictor
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command grand mean centers predictors in the Bayesian output, saved imputations are on the original metric
- ❖ Unspecified associations for predictor variables
- ❖ **NIMPS** command specifies 20 imputed data sets
- ❖ Setting **CHAINS** equal to **NIMPS** saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```

DATA: data1.dat;
VARIABLES: id v1 v2 v3 y x1 d x2 v4;
ORDINAL: d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL: y ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITER: 10000;
CHAINS: 20;
NIMPS: 20;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed.

```

VARIABLE ORDER IN IMPUTED DATA:

stacked = 'imps.dat'

imp# id n1 d o1 y x1 d2 x2 x3

```

The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp` to create multiple imputations and the `mitml` package (Grund, Robitzsch, & Lüdke, 2021) for analysis and pooling. Note that the `SAVE` command is no longer necessary because imputations are automatically stored in an `rblimp` list object called `mymodel@imputations`. The pooled estimates are numerically equivalent to the Bayesian results from Example 4.3.

```
library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  ordinal = 'd',
  fixed = 'd',
  center = 'x1 x2',
  model = 'y ~ x1 x2 d',
  seed = 90291,
  burn = 1000,
  iter = 10000,
  chains = 20,
  nimps = 20
)
output(mymodel)

# mitml list
implist <- as.mitml(mymodel)

# pooled grand means
mean_x1 <- mean(unlist(lapply(implist, function(data) mean(data$x1))))
mean_x2 <- mean(unlist(lapply(implist, function(data) mean(data$x2))))

# analysis and pooling with mitml
results <- with(implist, lm(y ~ I(x1 - mean_x1) + I(x2 - mean_x2) + d))
testEstimates(results, extra.pars = T, df.com = 626)
```

4.5: Linear Regression With Nominal Predictors

This example illustrates a linear regression model with a multicategorical nominal predictor. Clicking the links below downloads the scripts and data for this example,

and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.4a.imp](#) [Ex4.4b.imp](#) [data2.dat](#)

The regression model is

$$Y = \beta_0 + \beta_1 X^{cgm} + \beta_2 D + \beta_3 N_2 + \beta_3 N_3 + \beta_3 N_4 + \varepsilon_i$$

where Y is a continuous outcome, X is a continuous predictor, D is a dummy code, and N_2 , N_3 , and N_4 are dummy codes that represent a four-category nominal predictor ($N = 1, 2, 3, 4$). The *cgm* superscript denotes variables centered at their grand means. The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 4.4).

- ❖ `ORDINAL` command identifies a binary predictor
- ❖ `NOMINAL` command identifies a 4-category discrete predictor that Blimp automatically converts to dummy codes with the lowest numeric value as the reference group
- ❖ `FIXED` command identifies a complete predictor
- ❖ `CENTER` command grand mean centering to predictors
- ❖ Unspecified associations for predictor variables

```

DATA: data2.dat;
VARIABLES: id y v1 x d v2 n v3 v4;
ORDINAL: d;
NOMINAL: n;
MISSING: 999;
FIXED: x;
CENTER: x;
MODEL: y ~ x d n;
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rb1imp` script is as follows.

```

library(rblimp)
load(file = 'data2.rda')

mymodel <- rblimp(
  data = data2,
  ordinal = 'd',
  nominal = 'n',
  fixed = 'x',
  center = 'x',
  model = 'y ~ x d n',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

```

Alternatively, the individual dummy codes can be referenced on the MODEL line by appending their numeric code to the end of the predictor's name, as follows.

```

DATA: data2.dat;
VARIABLES: id y v1 x d v2 n v3 v4;
ORDINAL: d;
NOMINAL: n;
MISSING: 999;
FIXED: x;
CENTER: x;
MODEL: y ~ x d n.2 n.3 n.4;
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data2.rda')

mymodel <- rblimp(
  data = data2,
  ordinal = 'd',
  nominal = 'n',
  fixed = 'x',

```



```
center = 'x',  
model = 'y ~ x d n.2 n.3 n.4',  
seed = 90291,  
burn = 2000,  
iter = 10000  
)  
output(mymodel)
```

4.6: Fully Conditional Specification Multiple Imputation

The model-based multiple imputation procedure illustrated in Example 4.4 creates filled-in data sets tailored to the analysis specified on the `MODEL` line. The resulting imputations are appropriate for fitting the identical model (or one that is nested within the target model) in the frequentist framework. Fully conditional specification multiple imputation instead uses a round robin sequence of regression models, each of which features an incomplete variable regressed on all other variables (complete or previously imputed). Blimp's implementation of fully conditional specification is described in Chapter 2 (see the `FCS` command).

This example illustrates a fully conditional specification imputation routine that would yield appropriate imputations for the linear regression model from Example 4.5 (or any additive model that includes the variables listed on the `FCS` line). Note that fully conditional specification should not be applied to analysis models with interactive or nonlinear effects, as it is prone to bias in such cases (Bartlett et al., 2015; Seaman, Bartlett, & White, 2012). The model-based multiple imputation procedure illustrated in Example 4.8 is a better option. Clicking the links below downloads the scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.6.imp](#) [data2.dat](#)

The syntax highlights are as follows.

- ❖ `ORDINAL` command identifies binary variables
- ❖ `NOMINAL` command identifies a 4-category nominal variable

- ❖ **FIXED** command identifies complete variables
- ❖ **FCS** command includes all analysis variables plus two auxiliary variables
- ❖ **NIMPS** command specifies 20 imputed data sets
- ❖ Setting **CHAINS** equal to **NIMPS** saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```

DATA: data2.dat;
VARIABLES: id y1 y2 x1 d1 d2 n1 x2 n2;
ORDINAL: d1 d2;
NOMINAL: n1;
FIXED: x1 d2;
MISSING: 999;
FCS: y1 x1 d1 d2 n1 x2;
SEED: 90291;
BURN: 1000;
ITER: 1000;
CHAINS: 20;
NIMPS: 20;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed.

```
VARIABLE ORDER IN IMPUTED DATA:
```

```

stacked = 'imps.dat'

imp# id y1 y2 x1 d1 d2 n1 x2 n2

```

The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp_fcs` to create multiple imputations and the `mitml` package (Grund, Robitzsch, & Lüdke, 2021) for analysis and pooling. Note that the **MISSING** and

FCS commands are no longer necessary.. The former is omitted because that information is contained in the R data file. The FCS command is replaced by a `variables` parameter that lists the variables to be included in the imputation model. Additionally, the `SAVE` command is no longer necessary because imputations are automatically stored in an `rblimp` list object called `mymodel@imputations`.

```
library(rblimp)
load(file = 'data2.rda')

mymodel <- rblimp_fcs(
  data = data2,
  ordinal = 'd1 d2',
  nominal = 'n1',
  fixed = 'x1 d2',
  variables = 'y1 x1 d1 d2 n1 x2',
  seed = 90291,
  burn = 1000,
  iter = 1000,
  chains = 20,
  nimps = 20)

output(mymodel)

# mitml list
implist <- as.mitml(mymodel)

# pooled grand mean
mean_x1 <- mean(unlist(lapply(implist, function(data) mean(data$x1))))

# analysis and pooling with mitml
results <- with(implist, lm(y1 ~ I(x1 - mean_x1) + d1 + factor(n1)))
testEstimates(results, extra.pars = T, df.com = 1994)
```

4.7: Auxiliary Variables

This example illustrates how to add missing data auxiliary variables to a regression model. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.7a.imp](#) [Ex4.7b.imp](#) [data3.dat](#)

The model analysis model features a continuous variable and dummy code as predictors. The *cgm* superscript denotes variables centered at their grand means.

$$Y = \beta_0 + \beta_1 X^{cgm} + \beta_2 D + \varepsilon_i$$

In Blimp, auxiliary variables are introduced via a factored regression (sequential) specification where analysis variables predict the auxiliary variables and auxiliary variables predict each other in a cascading pattern (i.e., the first auxiliary predicts the second, the first and second predict the third, and so on).

$$A_1 = \gamma_{01} + \gamma_{11} Y + \gamma_{21} X^{cgm} + \gamma_{31} D + r_1$$

$$A_2 = \gamma_{02} + \gamma_{12} A_1 + \gamma_{22} Y + \gamma_{32} X^{cgm} + \gamma_{42} D + r_2$$

$$A_3 = \gamma_{03} + \gamma_{13} A_2 + \gamma_{23} A_1 + \gamma_{33} Y + \gamma_{43} X^{cgm} + \gamma_{53} D + r_3$$

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies binary variables
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean centering to a predictor
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command features a factored regression (sequential specification) for auxiliary variables
- ❖ Unspecified associations for predictor variables

```
DATA: data3.dat;
VARIABLES: id x a1 a2 y d a3 v1:v4;
MISSING: 999;
ORDINAL: d a3;
FIXED: d;
CENTER: x;
MODEL:
focal.model:
y ~ x d;
auxiliary.model:
```

```

a1 ~ y x d;
a2 ~ a1 y x d;
a3 ~ a1 a2 x d;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data3.rda')

mymodel <- rblimp(
  data = data3,
  ordinal = 'd a3',
  fixed = 'd',
  center = 'x',
  model = '
focal.model:
y ~ x d;
auxiliary.model:
a1 ~ y x d;
a2 ~ a1 y x d;
a3 ~ a1 a2 x d',
  seed = 90291,
  burn = 1000,
  iter = 10000
)
output(mymodel)

```

The script below illustrates a syntax shortcut that specifies the sequential specification by listing all auxiliary variables to the left of the tilde sign.

```

DATA: data3.dat;
VARIABLES: id x a1 a2 y d a3 v1:v4;
MISSING: 999;
ORDINAL: d a3;
FIXED: d;
CENTER: x;
MODEL:
focal.model:

```

```

y ~ x d;
auxiliary.model:
a3 a2 a1 ~ y x d;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

Adding the `NIMPS`, `CHAINS`, and `SAVE` commands to the script creates model-based multiple imputations that can be analyzed in the frequentist framework (see Example 4.4).

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data3.rda')

mymodel <- rblimp(
  data = data3,
  ordinal = 'd a3',
  fixed = 'd',
  center = 'x',
  model = '
focal.model:
y ~ x d;
auxiliary.model:
a3 a2 a1 ~ y x d',
  seed = 90291,
  burn = 1000,
  iter = 10000
)
output(mymodel)

```

4.8: Moderated Regression With an Interaction

This example illustrates a moderated regression with an interaction between a continuous predictor and binary moderator and an incomplete binary covariate. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.8a.imp](#) [Ex4.8b.imp](#) [Ex4.8b.R](#) [data4.dat](#)

The model is as follows, and the *cgm* superscript denotes variables centered at their grand means.

$$Y = \beta_0 + \beta_1 X^{cgm} + \beta_2 M + \beta_3 (X)(M) + \beta_4 D^{cgm} + \varepsilon$$

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies a binary predictor
- ❖ **NOMINAL** command identifies a binary predictor
- ❖ **FIXED** command identifies a complete variable
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ **MODEL** command features a product term
- ❖ **SIMPLE** command produces conditional effects (simple slopes) at each level of the nominal moderator
- ❖ Unspecified associations for predictor variables

```
DATA: data4.dat;
VARIABLES: id v1:v3 y x v4 v5 d m v6:v24;
ORDINAL: d;
NOMINAL: m;
MISSING: 999;
FIXED: m;
CENTER: x d;
MODEL: y ~ x m x*m d;
SIMPLE: x | m;
SEED: 90291;
BURN: 1000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
```

```

ordinal = 'd',
nominal = 'm',
fixed = 'm',
center = 'x d',
model = 'y ~ x m x*m d',
simple = 'x | m',
seed = 90291,
burn = 1000,
iter = 10000
)
output(mymodel)

```

Blimp can save multiple imputations from any model it estimates. The script below illustrates model-based multiple imputation (imputation tailored around one specific analysis) for the linear moderated regression model. The new syntax features are as follows.

- ❖ CENTER command grand mean centers predictors in the Bayesian output, but saved imputations are on the original metric
- ❖ NIMPS command specifies 20 imputed data sets
- ❖ Setting CHAINS equal to NIMPS saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```

DATA: data4.dat;
VARIABLES: id v1:v3 y x v4 v5 d m v6:v24;
ORDINAL: d;
NOMINAL: m;
MISSING: 999;
FIXED: m;
CENTER: x d;
MODEL: y ~ x m x*m d;
SIMPLE: x | m;
SEED: 90291;
BURN: 1000;
ITER: 10000;
CHAINS: 20;
NIMPS: 20;

```



```
SAVE: stacked = imps.dat;
```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed.

```
VARIABLE ORDER IN IMPUTED DATA:
```

```
stacked = 'imps.dat'
```

```
imp# id a1 a2 a3 y x1 x2 n1 d1 d2 o1 o2 o3 o4 o5 o6 o7 o8
      o9 o10 o11 o12 o13 o14 o15 o16 o17 o18 o19
```

The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp` to create multiple imputations and the `mitml` package (Grund, Robitzsch, & Lüdke, 2021) for analysis and pooling. Note that the `SAVE` command is no longer necessary because imputations are automatically stored in an `rblimp` list object called `mymodel@imputations`. The product term is not an imputed variable. Rather, the product is formed from the imputed lower-order variables, as shown below. The pooled multiple imputation estimates are numerically equivalent to the Bayesian results.

```
library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'd',
  nominal = 'm',
  fixed = 'm',
  center = 'x d',
  model = 'y ~ x m x*m d',
  simple = 'x | m',
  seed = 90291,
  burn = 1000,
```

```
  iter = 10000,
  chains = 20,
  nimps = 20
)
output(mymodel)

# mitml list
implist <- as.mitml(mymodel)

# pooled grand means
mean_x <- mean(unlist(lapply(implist, function(data) mean(data$x))))
mean_d <- mean(unlist(lapply(implist, function(data) mean(data$d))))

# analysis and pooling with mitml
results <- with(implist, lm(y ~ I(x - mean_x) + m + I(x - mean_x):m + I(d -
mean_d)))
testEstimates(results, extra.pars = T, df.com = 295)
```

4.9: Multiple Imputation Within Subgroups

Fully conditional specification multiple imputation is generally inappropriate for interactive effects because it is prone to bias. The moderated regression in Example 4.8 is an exception that could be handled by imputing the data separately within each group of the complete moderator variable (Enders & Gottschall, 2011; Graham, 2009). This example illustrates a multiple-group multiple imputation strategy that stratifies the data by subgroup and imputes within each strata. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.9.imp](#) [Ex4.9.R](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies a binary variable
- ❖ **FIXED** command identifies a complete variable
- ❖ **BYGROUP** identifies complete, nominal strata variable not listed on the **ORDINAL** (or **NOMINAL**) command

- ❖ FCS command includes all analysis variables (other than the one listed on the BYGROUP line) plus three auxiliary variables
- ❖ NIMPS command specifies 20 imputed data sets
- ❖ Setting CHAINS equal to NIMPS saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```

DATA: data4.dat;
VARIABLES: id a1:a3 y x v1 v2 d group v3:v21;
ORDINAL: d;
MISSING: 999;
FIXED: group;
BYGROUP: group;
FCS: a1:a3 y x d;
SEED: 90291;
BURN: 1000;
ITER: 10000;
CHAINS: 20;
NIMPS: 20;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed.

VARIABLE ORDER IN IMPUTED DATA:

```
stacked = 'imps.dat'
```

```

imp# id a1 a2 a3 y x v1 v2 d group v3 v4 v5 v6 v7 v8 v9
v10 v11 v12 v13 v14 v15 v16 v17 v18 v19 v20 v21

```

The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp_fcs` to create multiple imputations and the `mitml` package (Grund, Robitzsch, & Lüdke, 2021) for analysis and pooling. Note that the `MISSING` and

FCS commands are no longer necessary.. The former is omitted because that information is contained in the R data file. The FCS command is replaced by a `variables` parameter that lists the variables to be included in the imputation model. Additionally, the `SAVE` command is no longer necessary because imputations are automatically stored in an `rblimp` list object called `mymodel@imputations`. Finally, the `BYGROUP` command is replaced by listing `|> by_group('group')` after the `rblimp_fcs` function (`group` is the grouping variable's name in the data)

```
library(rblimp)
load(file = 'data2.rda')

mymodel <- rblimp_fcs(
  data = data4,
  ordinal = 'd',
  variables = 'a1:a3 y x d',
  seed = 90291,
  burn = 1000,
  iter = 10000,
  chains = 20,
  nimps = 20
) |> by_group('group')

lapply(mymodel,output)

# mitml list
implist <- as.mitml(mymodel)

# pooled grand means
mean_x <- mean(unlist(lapply(implist, function(data) mean(data$x))))
mean_d <- mean(unlist(lapply(implist, function(data) mean(data$d))))

# analysis and pooling with mitml
results <- with(implist,
  lm(y ~ I(x - mean_x) + group + I(x - mean_x):group + I(d - mean_d)))
testEstimates(results, extra.pars = T, df.com = 295)
```

4.10: Curvilinear Regression

This example illustrates a curvilinear regression with a quadratic term and continuous and binary covariates. Clicking the links below downloads the Blimp

scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.10.imp](#) [data5.dat](#)

The regression model is as follows, and the *cgm* superscript denotes variables centered at their grand means.

$$Y = \beta_0 + \beta_1 X_1^{cgm} + \beta_2 (X_1^{cgm})^2 + \beta_3 X_2^{cgm} + \beta_4 D_1 + \beta_5 D_2 + \varepsilon$$

The syntax highlights are listed below. Adding the **NIMPS** and **SAVE** commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ **ORDINAL** command identifies binary predictors
- ❖ **FIXED** command identifies complete predictors
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ **MODEL** command features an embedded function that squares a predictor
- ❖ Unspecified associations for predictor variables

```
DATA: data5.dat;
VARIABLES: id d1 d2 v1:v3 x1 x2 y;
MISSING: 999;
ORDINAL: d1 d2;
FIXED: d1 x2;
CENTER: x1 x2;
MODEL: y2 ~ x1 (x1^2) x2 d1 d2;
SEED: 12345;
BURN: 1000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data5.rda')

mymodel <- rblimp(
```

```

data = data5,
ordinal = 'd1 d2',
fixed = 'd1 x2',
center = 'x1 x2',
model = 'y ~ x1 (x1^2) x2 d1 d2',
seed = 12345,
burn = 1000,
iter = 10000
)
output(mymodel)

```

4.11: Probit Regression With a Binary Outcome

This example illustrates probit regression for a binary outcome. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.11a.imp](#) [Ex4.11b.imp](#) [data1.dat](#)

The model features a latent response variable regressed on continuous predictors and a binary dummy code, and the *cgm* superscript denotes variables centered at their grand means.

$$Y^* = \beta_0 + \beta_1 X_1^{cgm} + \beta_2 X_2^{cgm} + \beta_3 D + \varepsilon$$

A single threshold value fixed at zero is automatically included and does not require specification. The syntax highlights are listed below. Adding the **NIMPS** and **SAVE** commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ **ORDINAL** command identifies a binary outcome and predictor
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables

DATA: data1.dat;

```

VARIABLES: id v1 y v2 x1 x2 d v3 v4;
ORDINAL: y d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL:
y ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  ordinal = 'y d',
  fixed = 'd',
  center = 'x1 x2',
  model = 'y ~ x1 x2 d',
  seed = 90291,
  burn = 1000,
  iter = 10000
)
output(mymodel)

```

Blimp can also create auxiliary parameters that are functions of the estimated model parameters. To illustrate, the following script uses parameter labels, built-in functions, and the `PARAMETERS` command to compute the predicted probability of a “success” or “case” at each level of the D_1 dummy code (and at the means of the continuous predictors). The additional syntax highlights are as follows.

- ❖ `MODEL` command labels the intercept and the binary predictor’s slope
- ❖ `PARAMETERS` command defines news parameters that give the predicted probability of a “success” (outcome = 1) at each level of the dummy code and the group difference on the probability metric

```

DATA: data1.dat;
VARIABLES: id v1 y v2 x1 x2 d v3 v4;
ORDINAL: y d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL:
y ~ 1@b0 x1 x2 d@b3;
PARAMETERS:
pp_d0 = phi(b0);
pp_d1 = phi(b0 + b3);
pp_diff = pp_d1 - pp_d0;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  ordinal = 'y d',
  fixed = 'd',
  center = 'x1 x2',
  model = 'y ~ 1@b0 x1 x2 d@b3',
  parameters = 'pp_d0 = phi(b0);
pp_d1 = phi(b0 + b3);
pp_diff = pp_d1 - pp_d0',
  seed = 90291,
  burn = 1000,
  iter = 10000
)
output(mymodel)

```

4.12: Probit Regression With an Ordinal Outcome

This example illustrates a probit regression for an ordered categorical outcome with seven response options (e.g., a Likert scale). Clicking the links below downloads the

Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.12.imp](#) [data1.dat](#)

The model features a latent response variable regressed on continuous predictors and a binary dummy code, and the *cgm* superscript denotes variables centered at their grand means.

$$Y^* = \beta_0 + \beta_1 X_1^{cgm} + \beta_2 X_2^{cgm} + \beta_3 D + \varepsilon$$

Six threshold parameters that divide the latent response distribution into seven bins are automatically included and do not require specification (the lowest is fixed at zero for identification). The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ `ORDINAL` command identifies an ordinal outcome and a binary predictor
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ `FIXED` command identifies a complete predictor
- ❖ `CENTER` command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables
- ❖ Longer burn-in period for ordered categorical variables

```

DATA: data1.dat;
VARIABLES: id v1 v2 y x1 x2 d v3 v4;
ORDINAL: y d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL:
y ~ x1 x2 d;
SEED: 90291;
BURN: 20000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  ordinal = 'y d',
  fixed = 'd',
  center = 'x1 x2',
  model = 'y ~ x1 x2 d',
  seed = 90291,
  burn = 20000,
  iter = 10000
)
output(mymodel)
```

4.13: Logistic Regression With a Binary Outcome

This example illustrates logistic regression for a binary outcome. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.13a.imp](#) [Ex4.13b.imp](#) [data1.dat](#)

The model features a binary outcome regressed on continuous predictors and a binary dummy code, and the *cgm* superscript denotes variables centered at their grand means.

$$\ln \left(\frac{Pr(Y = 1)}{1 - Pr(Y = 1)} \right) = \beta_0 + \beta_1 X_1^{cgm} + \beta_2 X_2^{cgm} + \beta_3 D$$

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8). When saving imputations, adding the `savepredicted` keyword to the `OPTIONS` command saves predicted probabilities (see Example 4.21).

- ❖ `ORDINAL` command identifies a binary outcome and predictor

- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ Applying the **logit** function to the dependent variable on the **MODEL** line requests a logit rather than probit link
- ❖ Unspecified associations for predictor variables

```
DATA: data1.dat;  
VARIABLES: id v1 y v2 x1 x2 d v3 v4;  
ORDINAL: y d;  
MISSING: 999;  
FIXED: d;  
CENTER: x1 x2;  
MODEL:  
logit(y) ~ x1 x2 d;  
SEED: 90291;  
BURN: 2000;  
ITER: 10000;
```

Note that the outcome variable must be coded as 0 and 1 when using the logit command in conjunction with **ORDINAL**. If the outcome has different codes (e.g., 1 and 2), either use **TRANSFORM** to recode the variable or use the **NOMINAL** command to identify the outcome as categorical.

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'data1.rda')  
  
mymodel <- rblimp(  
  data = data1,  
  ordinal = 'y d',  
  fixed = 'd',  
  center = 'x1 x2',  
  model = 'logit(y) ~ x1 x2 d',  
  seed = 90291,  
  burn = 2000,  
  iter = 10000  
)  
output(mymodel)
```

Blimp can also create auxiliary parameters that are functions of the estimated model parameters. To illustrate, the following script uses parameter labels, built-in functions, and the `PARAMETERS` command to compute the predicted probability of a “success” or “case” at each level of the D_1 dummy code (and at the means of the continuous predictors). The additional syntax highlights are as follows.

- ❖ `MODEL` command labels the intercept and the binary predictor’s slope
- ❖ `PARAMETERS` command defines news parameters that give the predicted probability of a “success” (outcome = 1) at each level of the dummy code and the group difference on the probability metric

```

DATA: data1.dat;
VARIABLES: id v1 y v2 x1 x2 d v3 v4;
ORDINAL: y d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL:
logit(y) ~ 1@b0 x1 x2 d@b3;
PARAMETERS:
pp_d0 = exp(b0) / (1 + exp(b0));
pp_d1 = exp(b0 + b3) / (1 + exp(b0 + b3));
pp_diff = pp_d1 - pp_d0;
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  ordinal = 'y d',
  fixed = 'd',
  center = 'x1 x2',
  model = 'logit(y) ~ 1@b0 x1 x2 d@b3',
  parameters = 'pp_d0 = exp(b0) / (1 + exp(b0));

```

```

pp_d1 = exp(b0 + b3) / (1 + exp(b0 + b3));
pp_diff = pp_d1 - pp_d0',
seed = 90291,
burn = 2000,
iter = 10000
)
output(mymodel)

```

4.14: Logistic Regression With a Multicategorical Outcome

This example illustrates logistic regression for a multicategorical outcome with three levels. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.14.imp](#) [data4.dat](#)

The model features a 3-category outcome ($Y = 1, 2, 3$) regressed on three continuous predictors, with the lowest numeric code (e.g., $Y = 1$) as the reference group. The *cgm* superscript denotes variables centered at their grand means.

$$\ln \left(\frac{Pr(Y = 2)}{Pr(Y = 1)} \right) = \beta_{02} + \beta_{12}X_1^{cgm} + \beta_{22}X_2^{cgm} + \beta_{32}X_3^{cgm}$$

$$\ln \left(\frac{Pr(Y = 3)}{Pr(Y = 1)} \right) = \beta_{03} + \beta_{13}X_1^{cgm} + \beta_{23}X_2^{cgm} + \beta_{33}X_3^{cgm}$$

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ `NOMINAL` command identifies a multicategorical outcome, which automatically invokes a logit link when the categorical variable is an outcome (applying the `logit` function to the dependent variable is optional)
- ❖ `FIXED` command identifies a complete predictor
- ❖ `CENTER` command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables

```
DATA: data4.dat;  
VARIABLES: id x1:x3 v1:v3 y v4:v24;  
NOMINAL: y;  
MISSING: 999;  
FIXED: x2 x3;  
CENTER: x1 x2 x3;  
MODEL: logit(y) ~ x1 x2 x3;  
SEED: 90291;  
BURN: 2000;  
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'data4.rda')  
  
mymodel <- rblimp(  
  data = data4,  
  nominal = 'y',  
  fixed = 'x2 x3',  
  center = 'x1 x2 x3',  
  model = 'logit(y) ~ x1 x2 x3',  
  seed = 90291,  
  burn = 2000,  
  iter = 10000  
)  
output(mymodel)
```

4.15: Count Regression

This example illustrates a regression analysis with a count outcome. Blimp uses the negative binomial regression model described by Asparouhov and Muthén (2021). The negative binomial (NB) model is similar to the Poisson model, but it incorporates an additional overdispersion term that accounts for heterogeneity among individuals with the same predicted score. The interpretation of the coefficients is the same as a Poisson regression (Coxe, West, & Aiken, 2009). Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.15.imp](#) [data22.dat](#)

The model features a pair of continuous predictors and a binary dummy code, as follows. The *cgm* superscript denotes variables centered at their grand means.

$$\ln(\hat{\mu}) = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \beta_3 X_1^{cgm} + \beta_4 X_2^{cgm}$$

The term inside the natural log is the predicted count given the constellation of predictors on the right side of the equation. The model parameters reflect changes on the natural log of the count. As noted earlier, the model incorporates an overdispersion parameter that accommodates heterogeneity among individuals with the same predicted value.

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies binary predictors
- ❖ **COUNT** command identifies a count outcome
- ❖ **FIXED** command identifies complete predictors
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables

```
DATA: data22.dat;
VARIABLES: id d1 x1 v1 d2 x2 v2 y v3 v4;
ORDINAL: d1 d2;
COUNT: y;
MISSING: 999;
FIXED: d1 x1;
CENTER: x1 x2;
MODEL: y ~ d1 d2 x1 x2;
SEED: 90291;
BURN: 5000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data22.rda')
```

```

mymodel <- rblimp(
  data = data22,
  ordinal = 'd1 d2',
  count = 'y',
  fixed = 'd1 x1',
  center = 'x1 x2',
  model = 'y ~ d1 d2 x1 x2',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```

4.16: Zero-Inflated Count Outcome

This example illustrates a regression analysis with a count outcome that features excessive zeros. Blimp uses the negative binomial regression model described by Asparouhov and Muthén (2021). The negative binomial (NB) model is similar to the Poisson model, but it incorporates an additional overdispersion term that accounts for heterogeneity among individuals with the same predicted score. The interpretation of the coefficients is the same as a Poisson regression (Coxe, West, & Aiken, 2009).

A two-part model like that from Olsen and Schafer (2001) is used for the zero inflation. The two-part model features a binary indicator Y_{bin} that equals zero if the count variable Y equals zero and one if Y is greater than zero. The binary indicator is the dependent variable in a probit regression model predicting whether the count is non-zero. In probit regression, the binary indicator appears as a normally distributed latent response variable. In the model below, the latent response (denoted by an asterisk superscript) is regressed on a pair of binary dummy codes and continuous predictors.

$$Y_{bin}^* = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \beta_3 X_1^{cgm} + \beta_4 X_2^{cgm} + \varepsilon$$

The *cgm* superscript denotes centering at their grand mean. The probit model includes a single threshold value that is automatically fixed at zero for identification. The residual variance is similarly fixed at one.

The second part of the model considers only the non-zero counts. The outcome is a recoded version of Y that is missing whenever Y (or Y_{bin}) equals zero. In the example below, the non-zero counts are regressed on a pair of binary dummy codes and continuous covariates.

$$\ln(\hat{\mu}) = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \beta_3 X_1^{cgm} + \beta_4 X_2^{cgm}$$

The *cgm* superscript denotes grand mean centering. The term inside the natural log is the predicted count given the constellation of predictors on the right side of the equation. The model parameters reflect changes on the natural log of the count. As noted earlier, the model incorporates an overdispersion parameter that accommodates heterogeneity among individuals with the same predicted value.

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.16.imp](#) [data22.dat](#)

The syntax highlights are listed below. Adding the **NIMPS** and **SAVE** commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ **ORDINAL** command identifies binary predictors and the binary outcome indicating whether the count was greater than zero
- ❖ **COUNT** command identifies a count outcome
- ❖ **TRANSFORM** command creates the variables for the two-part model
- ❖ **FIXED** command identifies complete predictors
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables

```
DATA: data22.dat;
VARIABLES: id d1 x1 v1 d2 x2 v2 ycnt v3 v4;
ORDINAL: d1 d2;
ORDINAL: d1 d2 ybin;
COUNT: y;
TRANSFORM:
y = missing(ycnt == 0, ycnt);
ybin = ifelse(ycnt == 0, 0, 1);
MISSING: 999;
FIXED: d1 x1;
CENTER: x1 x2;
MODEL:
y ~ d1 d2 x1 x2;
ybin ~ d1 d2 x1 x2;
SEED: 90291;
BURN: 5000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data22.rda')

mymodel <- rblimp(
  data = data22,
  ordinal = 'd1 d2',
  count = 'y',
  transform = 'y = missing(ycnt == 0, ycnt);
ybin = ifelse(ycnt == 0, 0, 1)',
  fixed = 'd1 x1',
  center = 'x1 x2',
  model = '
y ~ d1 d2 x1 x2;
ybin ~ d1 d2 x1 x2',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)
```

4.17: Scale Scores With Incomplete Item Responses

This example illustrates a regression analysis that features a 6-item sum (scale) score as the outcome, a 7-item sum score as a predictor, and two binary covariates. The ordered categorical (e.g., questionnaire) items that determine the sum are incomplete. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.17a.imp](#) [Ex4.17b.imp](#) [data4.dat](#)

The analysis model is

$$\begin{aligned}
 Y &= \beta_0 + \beta_1 X + \beta_2 D_1 + \beta_3 D_2 + \varepsilon \\
 &= \beta_0 + \beta_1 (X_1 + \dots + X_7) + \beta_2 D_1 + \beta_3 D_2 + \varepsilon
 \end{aligned}$$

where X is the scale (sum) score, and X_1 to X_7 are its ordinal components. It is important to treat missing data at the item level when analyzing incomplete composite scores, as doing so maximizes power and precision. This example illustrates the approach from Alacam, Du, Enders, and Keller (2023) and Enders (2022). It is important to reiterate that the scale score (represented as an embedded function) is essentially a random variable that depends on its constituent parts rather than a deterministic computation or passive imputation (see the Functions Embedded in Equations section from Chapter 2). The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ `ORDINAL` command identifies binary and ordinal variables
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ `MODEL` command features a syntax shortcut that creates a factored regression (sequential) specification for all predictors

- ❖ MODEL command features an embedded function that defines the sum of ordinal items as a predictor
- ❖ MODEL command defines the sum of ordinal items as a random variable

```

DATA: data4.dat;
VARIABLES: id v1:v3 y v4:v6 d1 d2
              v7:v12 x1:x7 v13:v18;
ORDINAL: x1:x7 d1 d2;
MISSING: 999;
MODEL:
focal.model:
xscale = x1+:x7; # define sum function
y ~ xscale d1 d2; # embedded sum variable as a predictor
predictor.model:
x1:x7 d1 d2 ~ 1;
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4a.rda')

mymodel <- rblimp(
  data = data4a,
  ordinal = 'x1:x7 d1 d2',
  model = '
focal.model:
xscale = x1+:x7;
y ~ xscale d1 d2;
predictor.model:
x1:x7 d1 d2 ~ 1',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```

The previous script used a composite score as the dependent variable but did not incorporate the dependent variable's component items into the model. Doing so

would improve precision because the items are strong correlates of the sum score. The code block below leverages item-level correlations by introducing five of the six outcome items as auxiliary variables (Eekhout et al., 2015). The component items are added using the same auxiliary variable approach from Example 4.7. The additional syntax highlights are as follows.

- ❖ **MODEL** command features a factored regression (sequential specification) for the dependent variable's scale score and its items
- ❖ All but one of the dependent variable's scale items are used as auxiliary variables (using all items induces linear dependencies)

```

DATA: data4.dat;
VARIABLES: id a1 a2 a3 yscale v zscale n1 d1 d2
  y1:y6 x1:x7 z1:z6;
ORDINAL: x1:x7 d1 d2;
MISSING: 999;
MODEL:
  focal.model:
  xscale = x1+:x7;
  yscale ~ xscale d1 d2;
  predictor.model:
  x1:x7 d1 d2 ~ 1;
  auxiliary.models:
  # sequential specification for y scale items
  y1:y5 ~ yscale;
SEED: 90291;
BURN: 20000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'y1:y5 x1:x7 d1 d2',
  model = '
  focal.model:

```

```

xscale = x1:+:x7;
yscale ~ xscale d1 d2;
predictor.model:
x1:x7 d1 d2 ~ 1;
auxiliary.models:
y1:y5 ~ yscale',
seed = 90291,
burn = 20000,
iter = 10000
)
output(mymodel)

```

4.18: Scale Score Interactions

This example illustrates a moderated regression with an interaction between a 7-item sum score predictor and binary moderator. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.18.imp](#) [data4.dat](#)

The analysis model is

$$\begin{aligned}
 Y &= \beta_0 + \beta_1 X + \beta_2 M + \beta_3 (M)(X) + \varepsilon \\
 &= \beta_0 + \beta_1 (X_1 + \dots + X_7) + \beta_2 M + \beta_3 (M) (X_1 + \dots + X_7) + \varepsilon
 \end{aligned}$$

where X is the scale (sum) score, and X_1 to X_7 are its ordinal components. It is important to treat missing data at the item level when analyzing incomplete composite scores, as doing so maximizes power and precision. This example extends the approach from Alacam, Du, Enders, and Keller (2023) to include interaction effects involving an incomplete sum score. Enders (2022) summarizes the approach. It is important to reiterate that the scale score (represented as an embedded function) is essentially a random variable that depends on its constituent parts rather than a deterministic computation or passive imputation (see the Functions Embedded in Equations section from Chapter 2). The syntax highlights are listed

below. Adding the NIMPS and SAVE commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ ORDINAL command identifies binary and ordinal variables
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command features a syntax shortcut that creates a factored regression (sequential) specification for all predictors
- ❖ MODEL command features an embedded function that defines the sum of ordinal items as a predictor
- ❖ MODEL command defines the sum of ordinal items as a random variable
- ❖ MODEL command features a factored regression (sequential specification) for the dependent variable's scale score and its items
- ❖ All but one of the dependent variable's scale items are used as auxiliary variables (using all items induces linear dependencies)
- ❖ Longer burn-in period for estimating threshold parameters

```

DATA: data4.dat;
VARIABLES: id v1:v3 yscale xscale v4:v6 m
  y1:y6 x1:x7 v7:v12;
ORDINAL: y1:y5 x1:x7 m;
MISSING: 999;
MODEL:
  focal.model:
  xscale = x1:+:x7;
  yscale ~ xscale m m*xscale;
  predictor.model:
  x1:x7 m ~ 1;
  auxiliary.model:
  # sequential specification for y scale items
  y1:y5 ~ yscale;
SEED: 90291;
BURN: 20000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'y1:y5 x1:x7 m',
  model = '
  focal.model:
  xscale = x1:+:x7;
  yscale ~ xscale m m*xscale;
  predictor.model:
  x1:x7 m ~ 1;
  auxiliary.model:
  y1:y5 ~ yscale',
  seed = 90291,
  burn = 20000,
  iter = 10000
)
output(mymodel)

```

4.19: Skewed Predictor With a Yeo-Johnson Transformation

This example illustrates a Yeo-Johnson (Yeo & Johnson, 2000) transformation that samples imputations from a skewed distribution. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.19.imp](#) [data6.dat](#)

The analysis model is a logistic regression with two continuous variables and two binary dummy codes as predictors.

$$\ln \left(\frac{\Pr(Y = 1)}{1 - \Pr(Y = 1)} \right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 D_1 + \beta_4 D_2$$

X_2 's distribution is markedly peaked and positively skewed, and drawing imputations from a normal distribution would likely distort the variable's distribution.

The Yeo-Johnson procedure estimates the variable's shape and draws imputations from a nonnormal distribution. Applying the Yeo-Johnson transformation normalizes

the predictor variable, such that the resulting linear regression reflects associations between the normalized variable and other predictors. The sequential specification in the code block below invokes the following regression equation for the normalized predictor.

$$\tilde{X}_2 = \gamma_0 + \gamma_1 X_1 + \gamma_2 D_1 + r$$

However, skewed imputations on the raw score metric always appear on the right side of any regression equation (e.g., the focal regression model). Normalized imputations can be saved by adding the `saveLatent` keyword to the `OPTIONS` line.

The Yeo–Johnson transformation can be very slow (or fail) to converge if the skewed variable's mean is far from zero. To facilitate interpretation, the code block below centers the predictor scores at the median value of 16. Additional details about the procedure are available in the literature (Enders, 2022; Lüdtke et al., 2020b). The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 4.8).

- ❖ `ORDINAL` command identifies a binary outcome and predictors
- ❖ `FIXED` command identifies complete predictors
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ `MODEL` command features a factored regression (sequential specification) for incomplete predictor variables
- ❖ Unspecified associations for complete predictor variables
- ❖ Applying the `yjt` function to the skewed predictor on the `MODEL` line requests a Yeo-Johnson transformation
- ❖ Applying a subtraction function to center the skewed predictor at its median facilitates convergence
- ❖ Applying the `logit` function to the dependent variable on the `MODEL` line requests a logit rather than probit link

DATA: data6.dat;

```

VARIABLES: id d1 x1 v1 d2 v2 x2 v3 v4 y;
ORDINAL: y d1 d2;
MISSING: 999;
FIXED: d1 x1;
MODEL:
focal.model:
logit(y) ~ x1 x2 d1 d2;
predictor.model:
yjt(x2 - 16) ~ x1 d1;
d2 ~ x2 x1 d1;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data6.rda')

mymodel <- rblimp(
  data = data6,
  ordinal = 'y d1 d2',
  fixed = 'd1 x1',
  model = '
  focal.model:
logit(y) ~ x1 x2 d1 d2;
predictor.model:
yjt(x2 - 16) ~ x1 d1;
d2 ~ x2 x1 d1',
  seed = 90291,
  burn = 1000,
  iter = 10000
)
output(mymodel)

```

4.20: Skewed Outcome With a Yeo-Johnson Transformation

This example applies the Yeo–Johnson transformation to a nonnormal dependent variable. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.20a.imp](#) [Ex4.20b.imp](#) [Ex4.20.R](#) [data2.dat](#)

The untransformed analysis model features two continuous variables and one binary dummy code as predictors, where the *cgm* superscript denotes variables centered at their grand means.

$$Y = \beta_0 + \beta_1 X_1^{cgm} + \beta_2 X_2^{cgm} + \beta_3 D + \varepsilon$$

The outcome variable's distribution is markedly peaked and positively skewed. Applying the Yeo-Johnson transformation normalizes the dependent variable, such that the resulting linear regression reflects associations between the normalized outcome and the predictors.

$$\tilde{Y} = \gamma_0 + \gamma_1 X_1^{cgm} + \gamma_2 X_2^{cgm} + \gamma_3 D + r$$

Normalized imputations can be saved by adding the `saveLatent` keyword to the `OPTIONS` line. The Yeo-Johnson transformation can be very slow (or fail) to converge if the skewed variable's mean is far from zero. To facilitate interpretation, the code block below centers the outcome at the median value of 9. Additional details about the procedure are available in the literature (Enders, in press; Lüdtke et al., 2020b).

The syntax highlights are listed below.

- ❖ `ORDINAL` command identifies a binary predictor
- ❖ `FIXED` command identifies complete predictors
- ❖ Applying `yjt` function to the skewed outcome on the `MODEL` line requests a Yeo-Johnson transformation
- ❖ Applying a subtraction function to center the skewed outcome at its median facilitates convergence
- ❖ Unspecified associations for predictor variables

```
DATA: data2.dat;
VARIABLES: id y v1 x1 d v2 v3 x2 v4;
ORDINAL: d;
MISSING: 999;
```

```

FIXED: x1;
CENTER: x1 x2;
MODEL:
yjt(y - 9) ~ x1 x2 d;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data2.rda')

mymodel <- rblimp(
  data = data2,
  ordinal = 'd',
  fixed = 'x1',
  center = 'x1 x2',
  model = 'yjt(y - 9) ~ x1 x2 d',
  seed = 90291,
  burn = 1000,
  iter = 10000
)
output(mymodel)

```

Blimp can save multiple imputations from any model it estimates. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis, and listing the `saveLatent` keyword on the `OPTIONS` command saves the normalized imputes from the Yeo-Johnson transformation alongside the skewed imputes on the raw score metric (this keyword also saves the latent response scores for the binary predictor).

```

DATA: data2.dat;
VARIABLES: id y n1 x1 d1 d2 n2 x2 n3;
ORDINAL: d1;
MISSING: 999;
FIXED: x1;
CENTER: x1 x2;
MODEL:
yjt(y - 9) ~ x1 x2 d1;

```

```

SEED: 90291;
BURN: 3000;
ITER: 10000;
# save model-based multiple imputations;
CHAINS: 20;
NIMPS: 20;
OPTIONS: savelatent;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed.

VARIABLE ORDER IN IMPUTED DATA:

```

stacked = 'imps.dat'
imp# id y v1 x1 d v2 v3 x2 v4 yjt(yjt(y-9)) d.latent

```

The variable `y` contains skewed imputations on the raw score metric, and the variable `yjt(yjt(y-9))` contains the normalized imputes. The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp` to create multiple imputations and the `mitml` package (Grund, Robitzsch, & Lüdke, 2021) for analysis and pooling. Note that the `SAVE` command is no longer necessary because imputations are automatically stored in an `rblimp` list object called `mymodel@imputations`.

```

library(rblimp)
library(mitml)
load(file = 'data2.rda')

mymodel <- rblimp(
  data = data2,
  ordinal = 'd',
  fixed = 'x1',
  center = 'x1 x2',
  model = 'yjt(y - 9) ~ x1 x2 d',

```

```
seed = 90291,
burn = 1000,
iter = 10000,
chains = 20,
nimps = 20
)
output(mymodel)

# inspect variable names
names(mymodel@imputations[[1]])

# mitml list
implist <- as.mitml(mymodel)

# plot raw and transformed scores
dat2plot <- do.call(rbind, implist)
hist(dat2plot$y,breaks = 20)
hist(dat2plot$yjt.yjt.y.9..,breaks = 20)

# pooled grand means
mean_x1 <- mean(unlist(lapply(implist, function(data) mean(data$x1))))
mean_x2 <- mean(unlist(lapply(implist, function(data) mean(data$x2))))

# analyze skewed outcome
results <- with(implist,
  lm(y ~ I(x1 - mean_x1) + I(x2 - mean_x2) + d))
testEstimates(results, extra.pars = T, df.com = 1996)

# analyze normalized outcome
results <- with(implist,
  lm(yjt.yjt.y.9.. ~ I(x1 - mean_x1) + I(x2 - mean_x2) + d))
testEstimates(results, extra.pars = T, df.com = 1996)
```

To illustrate, the script below uses the R package `mitml` (Grund et al., 2021) to fit the regression model to the filled-in data sets. The positively skewed raw score

imputations are on the original metric, whereas the transformed imputations are approximately normal.

```
# set working directory
fdir::set()

# read data from working directory
imps <- read.table("imps.dat")
names(imps) <- c("imputation", "id", "y", "v1", "x1", "d",
               "v2", "v3", "x2", "v4", "ytransform", "d1.latent")

# plot raw and transformed scores
hist(imps$y)
hist(imps$ytransform)

# center predictors
imps$x1_cgm <- imps$x1 - mean(imps$x1)
imps$x2_cgm <- imps$x2 - mean(imps$x2)

# analysis and pooling with mitml
implist <- mitml::as.mitml.list(split(imps, imps$imputation))

# analyze skewed outcome
results <- with(implist, lm(y ~ x1_cgm + x2_cgm + d))
mitml::testEstimates(results, extra.pars = T, df.com = 1996)

# analyze transformed outcome
results <- with(implist, lm(ytransform ~ x1_cgm + x2_cgm + d))
mitml::testEstimates(results, extra.pars = T, df.com = 1996)
```

4.21: Propensity Score Estimation With Missing Data

This example illustrates propensity score estimation with missing data. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.21.imp](#) [data4.dat](#)

The focal model features a binary dummy code (the “treatment” indicator) predicting a continuous outcome.

$$Y = \beta_0 + \beta_1 D + \varepsilon$$

The propensity score model features the treatment indicator regressed on potential confounder variables and their higher-order interaction terms.

$$\ln\left(\frac{\Pr(D=1)}{1-\Pr(D=1)}\right) = \gamma_0 + \gamma_1 X_1 + \gamma_2 X_2 + \gamma_3 X_3 + \gamma_4 X_4 + \\ \gamma_5 X_1 X_2 + \gamma_6 X_1 X_3 + \gamma_7 X_1 X_4 + \gamma_8 X_2 X_3 + \gamma_9 X_2 X_4 + \gamma_{10} X_3 X_4$$

Because the treatment indicator D_1 consists of naturally occurring groups, this variable could be incomplete, which it is here. In this case, it is important for propensity score estimation to account for both models.

- ❖ **ORDINAL** command identifies a binary outcome and predictor
- ❖ **FIXED** command identifies a complete predictor
- ❖ Applying the **logit** function to the dependent variable on the **MODEL** line requests a logit rather than probit link
- ❖ **FIXED** command identifies complete predictors
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command features a product term
- ❖ The **savepredicted** keyword on the **OPTIONS** line saves the predicted probabilities of treatment group membership, which are the propensity scores
- ❖ Unspecified associations for predictor variables
- ❖ **NIMPS** command specifies 20 imputed data sets
- ❖ Setting **CHAINS** equal to **NIMPS** saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```
DATA: data4.dat;
VARIABLES: id x1:x4 y v1 v2 d v3:v22;
ORDINAL: d;
```



```

MISSING: 999;
FIXED: x2 x3;
MODEL:
focal.model:
y ~ d;
propensity.model:
logit(d) ~ x1 x2 x3 x4 x1*x2 x1*x3 x1*x4 x2*x3 x2*x4 x3*x4;
SEED: 90291;
BURN: 2000;
ITER: 10000;
OPTIONS: savepredicted;
CHAINS: 20;
NIMPS: 20;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed.

VARIABLE ORDER IN IMPUTED DATA:

```

stacked = 'imps.dat'

imp# id x1 x2 x3 x4 y v1 v2 d v3 v4 v5 v6 v7 v8 v9 v10
v11 v12 v13 v14 v15 v16 v17 v18 v19 v20 v21 v22
y.predicted d.predicted

```

The variable `d.predicted` contains the propensity scores. Following earlier examples, imputed data sets from Blimp can be analyzed in other software packages.

The corresponding `rblimp` script is as follows. Note that the `SAVE` command is no longer necessary because imputations are automatically stored in a `rblimp` list object called `mymodel@imputations`. Predicted probabilities are automatically stored in a `rblimp` list object called `mymodel@predicted`.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(

```

```

data = data4,
ordinal = 'd',
fixed = 'x2 x3',
model = '
focal.model:
y ~ d;
propensity.model:
logit(d) ~ x1 x2 x3 x4 x1*x2 x1*x3 x1*x4 x2*x3 x2*x4 x3*x4',
seed = 90291,
burn = 2000,
iter = 10000,
chains = 20,
nimps = 20
)
output(mymodel)

```

4.22: Sampling Weights

This example illustrates a linear regression analysis with sampling (i.e., inverse probability) weights. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.22.imp](#) [data21.dat](#)

The analysis model features three continuous predictors.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \varepsilon$$

Blimp's MCMC estimation routine incorporates sampling weights using the procedure described in Goldstein (2011, Section 3.4.2). The syntax highlights are as follows.

- ❖ **WEIGHT** command identifies the inverse probability sampling weights
- ❖ Unspecified associations for predictor variables

```

DATA: data21.dat;
VARIABLES: v1:v3 wght y x1:x3 v4 v5;

```

```
WEIGHT: wght;  
MISSING: 999;  
MODEL: y ~ x1 x2 x3;  
SEED: 90291;  
BURN: 1000;  
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'data21.rda')  
  
mymodel <- rblimp(  
  data = data21,  
  model = 'y ~ x1 x2 x3',  
  weights = 'wght',  
  seed = 90291,  
  burn = 1000,  
  iter = 10000  
)  
output(mymodel)
```

4.23: Wald Significance Tests

This example illustrates the linear regression analysis from Example 4.3 with the Bayesian Wald test described by Asparouhov and Muthén (2021). These tests are printed by default for selected individual parameters. This example illustrates how to specify a custom significance test for evaluating the omnibus null hypothesis that all regression slopes equal zero. Other tests are possible as well (e.g., tests of equality constraints, nested model tests). Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex4.23.imp](#) [data1.dat](#)

The model features a pair of continuous predictors and a binary dummy code, where the *cgm* superscript denotes variables centered at their grand means.

$$Y = \beta_0 + \beta_1 X_1^{cgm} + \beta_2 X_2^{cgm} + \beta_3 D + \varepsilon$$

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies a binary predictor
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ **WALDTEST** commands specify three custom Bayesian Wald significance tests
- ❖ Unspecified associations for predictor variables

```
DATA: data1.dat;
VARIABLES: id v1:v3 y x1 d x2 v4;
ORDINAL: d;
MISSING: 999;
FIXED: d;
CENTER: x1 x2;
MODEL: y ~ x1@b1 x2@b2 d@b3;
WALDTEST: b1:b3 = 0;
WALDTEST: b1:b2 = 0;
WALDTEST: b1 = b2;
SEED: 90291;
BURN: 1000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows. Notice that each test is an element in a list.

```
library(rblimp)
load(file = 'data21.rda')

mymodel <- rblimp(
  data = data1,
  ordinal = 'd',
  fixed = 'd',
  center = 'x1 x2',
  model = 'y ~ x1@b1 x2@b2 d@b3',
  waldtest = list('b1:b3 = 0', 'b1:b2 = 0', 'b1 = b2'),
  seed = 90291,
  burn = 1000,
```

```
iter = 10000)
output(mymodel)
```

The `WALDTEST` command produces the output table below. The Wald test statistic is a chi-square variable, and the test's degrees of freedom equals the number of parameters by which the two models differ. The chi-square is an MCMC-generated estimate of a frequentist test statistic, and the p -value is the area above the test statistic value in a chi-square distribution. As such, the test can be used for frequentist inference. This approach adopts MCMC estimation for its computational benefits rather than its philosophical appeal (i.e., Bayes as computational frequentism; Levy & McNeish, 2023). By default, Blimp prints univariate Wald tests for all parameters except variances and variance explained test statistics.

WALD TESTS (Asparouhov & Muthén, 2021)

Test #1

Full:

```
[1] y ~ Intercept x1@b1 x2@b2 d@b3
```

Restricted:

```
[1] y ~ Intercept x1@b1 x2@b2 d@b3
```

Constraints in Restricted:

```
[1] b1 = 0
```

```
[2] b2 = 0
```

```
[3] b3 = 0
```

Wald Statistic (Chi-Square)	158.084
Number of Parameters Tested (df)	3
Probability	0.000

Test #2

Full:

```
[1] y ~ Intercept x1@b1 x2@b2 d@b3
```

Restricted:

```
[1] y ~ Intercept x1@b1 x2@b2 d@b3
```

Constraints in Restricted:

```
[1] b1 = 0
```

```
[2] b2 = 0
```

Wald Statistic (Chi-Square)	113.818
Number of Parameters Tested (df)	2
Probability	0.000

Test #3

Full:

```
[1] y ~ Intercept x1@b1 x2@b2 d@b3
```

Restricted:

```
[1] y ~ Intercept x1@b1 x2@b2 d@b3
```

Constraints in Restricted:

```
[1] b1 = b2
```

Wald Statistic (Chi-Square)	28.673
Number of Parameters Tested (df)	1
Probability	0.000

5 Path Analysis and Latent Variable Model Examples

This section illustrates path analyses and latent variable models in Blimp. These multivariate analyses are specified as collections of univariate equations. In general, it is possible to mix and match features from any examples to easily create complex analysis models that honor features of the data. Additional details about fitting path and latent variable models in Blimp can be found in Keller (2022), which is available for download [here](#).

Following the previous chapter, the examples in this section use a generic notation system where variable names usually consist of an alphanumeric prefix and a numeric suffix (e.g., Y , X_1 , X_2 , N_1 , D_1 , D_2). The letter Y designates a dependent variable, a D prefix denotes a binary dummy variable, an O prefix indicates an ordinal variable, and an N prefix indicates a multicategorical nominal variable. Finally, the model equations use a “cgm” superscript to indicate grand mean centering. The following list outlines the examples in this section.

- ❖ 5.1: Mediation Analysis
- ❖ 5.2: Moderated Mediation
- ❖ 5.3: Mediation With a Binary Outcome
- ❖ 5.4: Mediation With a Categorical Mediator
- ❖ 5.5: Mediation With a Count Outcome
- ❖ 5.6: Mediation With a Zero-Inflated Count Outcome
- ❖ 5.7: CFA With Continuous Indicators
- ❖ 5.8: CFA With Binary Indicators (2-Parameter IRT Model)
- ❖ 5.9: CFA With Ordinal Indicators
- ❖ 5.10: Imputing Latent Response Scores for Item-Level Factor Analysis
- ❖ 5.11: Skewed Indicators With a Yeo-Johnson Transformation
- ❖ 5.12: Latent Variable Regression Model
- ❖ 5.13: Latent-by-Manifest Variable Interaction
- ❖ 5.14: Moderated Nonlinear Factor Analysis (MNLFA)

- ❖ 5.15: Latent-by-Latent Variable Interaction
- ❖ 5.16: Three-Way Latent Variable Interaction
- ❖ 5.17: Latent Growth Curve Model

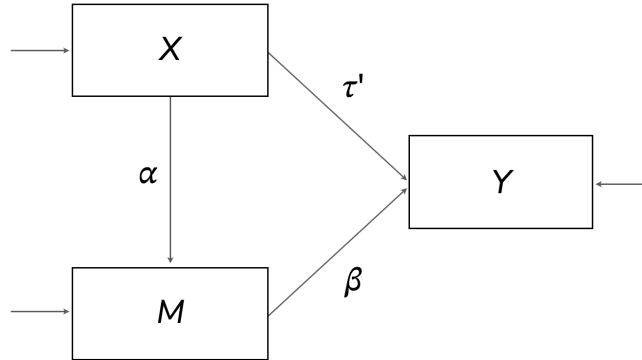
5.1: Mediation Analysis

This example illustrates a single-mediator path model. The regression models are shown below

$$M = I_M + \alpha X + \varepsilon_M$$

$$Y = I_Y + \beta M + \tau' X + \varepsilon_Y$$

where α and β are slope coefficients that define the indirect effect or product of the coefficients estimator, and τ' is the direct effect of X on Y . A path diagram of the analysis is shown below. The model also incorporates three auxiliary variables following the procedure from Example 4.7.



Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.1.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command labels the indirect effect's component pathways
- ❖ MODEL command features a syntax shortcut that creates a factored regression (sequential) specification for auxiliary variables
- ❖ PARAMETERS command uses labeled quantities to compute the product of coefficients estimator

```

DATA: data4.dat;
VARIABLES: id a1:a3 v1 y m v2 x v3:v22;
MISSING: 999;
MODEL:
mediation.model:
m ~ x@alpha;
y ~ m@beta x;
auxiliary.model:
a1:a3 ~ y m x;
PARAMETERS:
indirect = alpha * beta;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  model = '
mediation.model:
m ~ x@alpha;
y ~ m@beta x;
auxiliary.model:
a1:a3 ~ y m x',
  parameters = 'indirect = alpha * beta',
  seed = 90291,
  burn = 1000,
  iter = 10000

```

```
)  
output(mymodel)
```

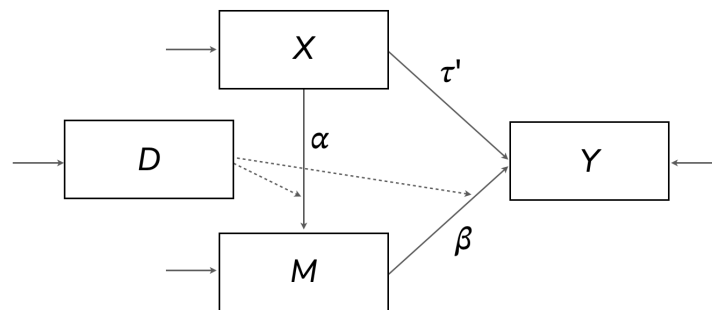
5.2: Moderated Mediation

This example adds moderated pathways to the single-mediator model from the previous example. The regression models are shown below

$$M = I_M + \alpha X + \gamma_1 D + \gamma_2 (X)(D) + \varepsilon_M$$

$$Y = I_Y + \beta M + \tau' X + \gamma_3 D + \gamma_4 (M)(D) + \varepsilon_Y$$

and the corresponding path diagram is as follows.



The dashed lines pointing from D to the directed arrows convey that D moderates the mediation model paths.

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.2.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ `ORDINAL` command identifies a binary predictor
- ❖ `FIXED` command identifies a complete predictor

- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command labels the indirect effect's component pathways
- ❖ MODEL command features a product term
- ❖ MODEL command features a syntax shortcut that creates a factored regression (sequential) specification for auxiliary variables
- ❖ PARAMETERS command uses labeled quantities to compute the product of coefficients estimator at each level of the binary moderator

```

DATA: data4.dat;
VARIABLES: id a1:a3 v1 y m v2 x d v3:v21;
ORDINAL: d;
MISSING: 999;
FIXED: d;
MODEL:
mediation.model:
m ~ x@alpha d x*d@alphamod;
y ~ m@beta x d m*d@betamod;
auxiliary.model:
# sequential specification for auxiliary variables
a1:a3 ~ y m x d;
PARAMETERS:
indirect.d0 = alpha * beta;
indirect.d1 = ( alpha + alphamod ) * ( beta + betamod );
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'd',
  fixed = 'd',
  model = '
mediation.model:
m ~ x@alpha d x*d@alphamod;

```

```

y ~ m@beta x d m*d@betamod;
auxiliary.model:
a1:a3 ~ y m x d',
parameters = 'indirect.d0 = alpha * beta;
indirect.d1 = ( alpha + alphamod ) * ( beta + betamod )',
seed = 90291,
burn = 1000,
iter = 10000
)
output(mymodel)

```

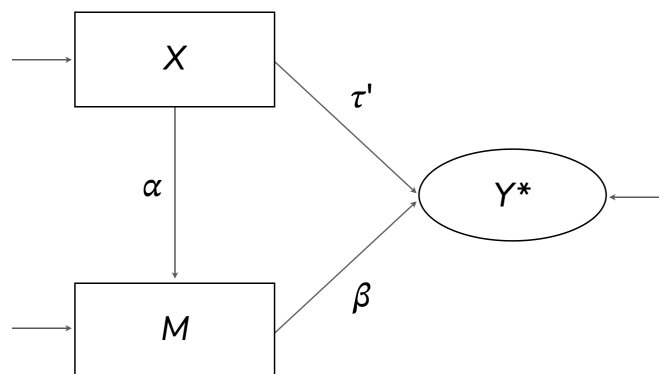
5.3: Mediation With a Binary Outcome

This example illustrates a single-mediator model with a binary outcome. The regression models are shown below

$$M = I_M + \alpha X + \varepsilon_M$$

$$Y^* = I_Y + \beta M + \tau' X + \varepsilon_{Y^*}$$

where Y^* denotes the underlying latent response variable for a binary outcome Y , and all other features of the model are the same as Example 5.1. A path diagram of the mediation model is shown below, with the ellipse denoting the latent response variable, the residual variance of which is fixed at one for identification.



Muthén, Muthén, and Asparouhov (2016) show that the indirect effect alone does not capture the fact that changes in Y are non-constant on the probability metric. They

give the following expression for computing the probability that $Y = 1$ given a particular value of X (Equations 8.4 to 8.6). The script below uses these equations to compute the probability at different values of X .

$$E(Y^*|X) = I_Y + I_M\beta + \alpha\beta X + \tau'X =$$

$$V(Y^*|X) = \beta^2\sigma_M^2 + 1$$

$$P(Y = 1|X) = \phi[E(Y^*|X)/\sqrt{V(Y^*|X)}]$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.3a.inp](#) [Ex5.3b.inp](#) [data20.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies a binary outcome and predictor
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command labels intercepts, slopes, and the residual variance of the mediator
- ❖ **MODEL** command features a syntax shortcut that creates a factored regression (sequential) specification for auxiliary variables
- ❖ **PARAMETERS** command uses labeled quantities to compute the product of coefficients estimator
- ❖ **PARAMETERS** command uses labeled quantities to compute the the probability that Y equals 1 at three values of X (the mean and plus/minus one standard deviation from the mean)

DATA: data20.dat;

```

VARIABLES: id a1:a3 v1 y m v2 x v3:v22;
MISSING: 999;
ORDINAL: y;
CENTER: x;
MODEL:
mediation.model:
# single-mediator model with parameter labels
m ~ 1@m_icept x@alpha;
m ~~ m@m_resvar;
y ~ 1@y_icept m@beta x@tau;
auxiliary.model:
a1:a3 ~ y m x;
PARAMETERS:
xvalue1 = -.50;
xvalue2 = 0;
xvalue3 = .50;
ab_xval1 = phi((y_icept + beta*m_icept +
  beta*alpha*xvalue1 + tau*xvalue1)/
  sqrt(beta^2*m_resvar + 1));
ab_xval2 = phi((y_icept + beta*m_icept +
  beta*alpha*xvalue2 + tau*xvalue2)/
  sqrt(beta^2*m_resvar + 1));
ab_xval3 = phi((y_icept + beta*m_icept +
  beta*alpha*xvalue3 + tau*xvalue3)/
  sqrt(beta^2*m_resvar + 1));
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data20.rda')

mymodel <- rblimp(
  data = data20,
  ordinal = 'y',
  center = 'x',
  model = '
mediation.model:
m ~ 1@m_icept x@alpha;
m ~~ m@m_resvar;
y ~ 1@y_icept m@beta x@tau;

```

```

auxiliary.model:
a1:a3 ~ y m x',
parameters = 'xvalue1 = -.50;
xvalue2 = 0;
xvalue3 = .50;
ab_xval1 = phi((y_icept + beta*m_icept +
  beta*alpha*xvalue1 + tau*xvalue1)/
  sqrt(beta^2*m_resvar + 1));
ab_xval2 = phi((y_icept + beta*m_icept +
  beta*alpha*xvalue2 + tau*xvalue2)/
  sqrt(beta^2*m_resvar + 1));
ab_xval3 = phi((y_icept + beta*m_icept +
  beta*alpha*xvalue3 + tau*xvalue3)/
  sqrt(beta^2*m_resvar + 1))',
seed = 90291,
burn = 1000,
iter = 10000
)
output(mymodel)

```

The script above defines the binary outcome as a latent response variable (i.e., probit regression). Applying the `logit` function to the dependent variable on the `MODEL` line requests a logit rather than probit link.

$$M = I_M + \alpha X + \varepsilon_M$$

$$\text{logit}(Y) = I_Y + \beta M + \tau' X$$

The conditional probabilities from Muthén, Muthén, and Asparouhov (2016) shown above do not have a simple expression when using a logit link. Instead, we can use the conditional indirect effects expressions defined in Geldhof et al. (2018, p. 304).

They define the conditional indirect effect at a given value of X as follows

$$\alpha\beta|X = \alpha \times \frac{\beta \times e^{I_Y + \beta M + \tau' X}}{(1 + e^{I_Y + \beta M + \tau' X})^2}$$

The `PARAMETERS` command in the script below computes these conditional indirect effects at three values of X .

```

DATA: data20.dat;
VARIABLES: id a1:a3 v1 y m v2 x v3:v22;
MISSING: 999;
ORDINAL: y;
CENTER: x;
MODEL:
mediation.model:
# single-mediator model with parameter labels
m ~ 1@m_icept x@alpha;
logit(y) ~ 1@y_icept m@beta x@tau;
auxiliary.model:
# sequential specification for auxiliary variables
a1:a3 ~ y m x;
PARAMETERS:
xvalue1 = -.50;
xvalue2 = 0;
xvalue3 = .50;
ab_xval1 = alpha *
  (beta*exp(y_icept + beta*m_icept + tau*xvalue1)) /
  (1 + exp(y_icept + beta*m_icept + tau*xvalue1))^2;
ab_xval2 = alpha *
  (beta*exp(y_icept + beta*m_icept + tau*xvalue2)) /
  (1 + exp(y_icept + beta*m_icept + tau*xvalue2))^2;
ab_xval3 = alpha *
  (beta*exp(y_icept + beta*m_icept + tau*xvalue3)) /
  (1 + exp(y_icept + beta*m_icept + tau*xvalue3))^2;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data20.rda')

mymodel <- rblimp(
  data = data20,
  ordinal = 'y',
  center = 'x',
  model = '
mediation.model:
m ~ 1@m_icept x@alpha;
logit(y) ~ 1@y_icept m@beta x@tau;

```



```

auxiliary.model:
a1:a3 ~ y m x',
parameters = 'xvalue1 = -.50;
xvalue2 = 0;
xvalue3 = .50;
ab_xval1 = alpha * (beta*exp(y_icept + beta*m_icept + tau*xvalue1)) /
  (1 + exp(y_icept + beta*m_icept + tau*xvalue1))^2;
ab_xval2 = alpha * (beta*exp(y_icept + beta*m_icept + tau*xvalue2)) /
  (1 + exp(y_icept + beta*m_icept + tau*xvalue2))^2;
ab_xval3 = alpha * (beta*exp(y_icept + beta*m_icept + tau*xvalue3)) /
  (1 + exp(y_icept + beta*m_icept + tau*xvalue3))^2',
seed = 90291,
burn = 1000,
iter = 10000
)
output(mymodel)

```

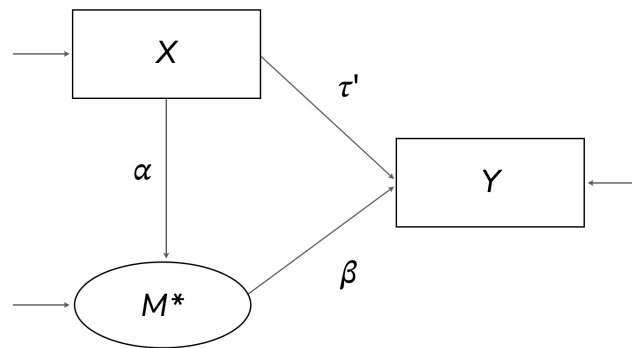
5.4: Mediation With a Categorical Mediator

This example illustrates a single-mediator path model with an ordered categorical mediator (the mediator could also be binary). The regression models are shown below

$$M^* = I_M + \alpha X + \varepsilon_M^*$$

$$Y = I_Y + \beta M^* + \tau' X + \varepsilon_Y$$

where α and β are slope coefficients that define the indirect effect or product of the coefficients estimator, and τ' is the direct effect of X on Y . A path diagram of the analysis is shown below. The model also incorporates three auxiliary variables following the procedure from Example 4.7.



When M is binary or ordinal, the α path represents the regression of a latent response variable on X . Typically, the discrete M would then serve as a predictor of Y , thus leading to an awkward situation where M essentially has two different metrics within the same model (i.e., M is latent when it is an outcome variable but ordinal when it is a predictor). Alternatively, Blimp can use the latent response variable in both regressions, effectively converting a complicated categorical variable regression into a straightforward linear regression with latent response variables. This idea was proposed in Muthén, Muthén, and Asparouhov (2016). Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.4a.imp](#) [Ex5.4b.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies an ordinal mediator
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command labels the indirect effect's component pathways
- ❖ **MODEL** command features a syntax shortcut that creates a factored regression (sequential) specification for auxiliary variables
- ❖ Appending the `.latent` suffix to the mediator's variable name in the **MODEL** statement accesses the latent response variable instead of the discrete responses

- ❖ **PARAMETERS** command uses labeled quantities to compute the product of coefficients estimator

```

DATA: data4.dat;
VARIABLES: id a1:a3 v1 y v2 v3 x v4:v22 m;
MISSING: 999;
ORDINAL: m;
MODEL:
mediation.model:
m ~ x@alpha;
y ~ m.latent@beta x;
auxiliary.model:
a1:a3 ~ y m.latent x;
PARAMETERS:
indirect = alpha * beta;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4a.rda')

mymodel <- rblimp(
  data = data4a,
  ordinal = 'm',
  center = 'x',
  model = '
mediation.model:
m ~ x@alpha;
y ~ m.latent@beta x;
auxiliary.model:
a1:a3 ~ y m.latent x',
  parameters = 'indirect = alpha * beta',
  seed = 90291,
  burn = 10000,
  iter = 10000
)
output(mymodel)

```

An alternate approach uses the conditional indirect effects expressions defined in Geldhof et al. (2018, p. 304). To illustrate, suppose that M is a binary mediator. The mediation model equations adopt a logistic model for M .

$$\text{logit}(M) = I_M + \alpha X$$

$$Y = I_Y + \beta M + \tau' X + \varepsilon_Y$$

Applying the `logit` function to the mediator variable on the `MODEL` line requests a logit link for this variable. Geldhof et al. define conditional indirect effects that reflect changes to M on the probability metric as follows.

$$\alpha\beta|X = \frac{\alpha \times e^{I_M + \alpha X}}{(1 + e^{I_M + \alpha X})^2} \times \beta$$

The `PARAMETERS` command in the script below computes these conditional indirect effects at three values of X . The `TRANSFORM` command uses an `ifelse` function to recode the 7-point M into a binary mediator.

```

DATA: data4.dat;
VARIABLES: id a1:a3 v1 y v2 v3 x v4:v22 m;
MISSING: 999;
TRANSFORM: m = ifelse(m7pt <= 4, 0, 1);
ORDINAL: m;
CENTER: x;
MODEL:
  mediation.model:
  logit(m) ~ 1@m_icept x@alpha;
  y ~ m@beta x;
  auxiliary.model:
  # sequential specification for auxiliary variables
  a1:a3 ~ y m x;
PARAMETERS:
  xvalue1 = -.50;
  xvalue2 = 0;
  xvalue3 = .50;
  ab_xval1 = (alpha*exp(m_icept + alpha*xvalue1)) /
    (1 + exp(m_icept + alpha*xvalue1))^2 * beta;
  ab_xval2 = (alpha*exp(m_icept + alpha*xvalue2)) /
    (1 + exp(m_icept + alpha*xvalue2))^2 * beta;

```

```

ab_xval3 = (alpha*exp(m_icept + alpha*xvalue3)) /
  (1 + exp(m_icept + alpha*xvalue3))^2 * beta;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'm',
  transform = 'm = ifelse(m7pt <= 4, 0, 1)',
  center = 'x',
  model = '
mediation.model:
logit(m) ~ 1@m_icept x@alpha;
y ~ m@beta x;
auxiliary.model:
a1:a3 ~ y m x',
  parameters = 'xvalue1 = -.50;
xvalue2 = 0;
xvalue3 = .50;
ab_xval1 = (alpha*exp(m_icept + alpha*xvalue1)) /
  (1 + exp(m_icept + alpha*xvalue1))^2 * beta;
ab_xval2 = (alpha*exp(m_icept + alpha*xvalue2)) /
  (1 + exp(m_icept + alpha*xvalue2))^2 * beta;
ab_xval3 = (alpha*exp(m_icept + alpha*xvalue3)) /
  (1 + exp(m_icept + alpha*xvalue3))^2 * beta',
  seed = 90291,
  burn = 10000,
  iter = 10000,
  output = 'default wald pvalue')

```

5.5: Mediation With a Count Outcome

This example illustrates a single-mediator model with a count outcome. Additional details about Blimp's count (negative binomial) modeling procedure is found in

Examples 4.15 and 4.16. The model equations are shown below, where X is binary and M is a continuous mediator.

$$M = I_M + \alpha X + \varepsilon_M$$

$$\ln(\hat{\mu}) = I_Y + \beta M + \tau' X$$

The term inside the natural log is the predicted count given the constellation of predictors on the right side of the equation. The β and τ' coefficients reflect changes on the natural log of the count. As noted in Example 4.15, the model incorporates an overdispersion parameter that accommodates heterogeneity among individuals with the same predicted value.

Geldhof et al. (2018) show that indirect effects that reflect changes to Y on the count metric are conditional on the values of X . In this example, X is binary, so there are two conditional indirect effects. They define the conditional indirect effect at a given value of X as follows

$$\alpha\beta|X = \alpha \times (\beta e^{I_Y + \beta M + \tau' X})$$

The `PARAMETERS` command in the script below computes these conditional indirect effects at each value of X .

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.5.imp](#) [data22.dat](#)

The syntax highlights are as follows.

- ❖ `ORDINAL` command identifies binary predictors
- ❖ `COUNT` command identifies a count outcome
- ❖ `MODEL` command labels the indirect effect's component pathways
- ❖ `PARAMETERS` command uses labeled quantities to compute conditional indirect effects

```

DATA: data22.dat;
VARIABLES: id x v1:v4 m y v5 v6;
ORDINAL: x;
COUNT: y;
MISSING: 999;
MODEL:
m ~ 1@m_icept x@alpha;
y ~ 1@y_icept m@beta x@tau;
PARAMETERS:
x0 = 0;
x1 = 1;
ab_at_x0 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x0));
ab_at_x1 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x1));
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data22.rda')

mymodel <- rblimp(
  data = data22,
  ordinal = 'x',
  count = 'y',
  model = '
m ~ 1@m_icept x@alpha;
y ~ 1@y_icept m@beta x@tau',
  parameters = 'x0 = 0;
x1 = 1;
ab_at_x0 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x0));
ab_at_x1 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x1))',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```

5.6: Mediation With a Zero-Inflated Count Outcome

This example illustrates a single-mediator model with a count outcome with excessive zeros. The procedure for the example is described in O'Rourke and Han (2023). Additional details about Blimp's count (negative binomial) modeling procedure is found in Examples 4.15 and 4.16.

A two-part model like that from Olsen and Schafer (2001) is used for the zero inflation. The two-part model features a binary indicator Y_{bin} that equals zero if the count variable Y equals zero and one if Y is greater than zero. The binary indicator is the dependent variable in a probit regression model predicting whether the count is non-zero. In probit regression, the binary indicator appears as a normally distributed latent response variable. In the model below, the latent response (denoted by an asterisk superscript) is regressed on a pair of binary dummy codes and continuous predictors.

$$Y_{bin}^* = \gamma_0 + \gamma_1 X + \gamma_2 M + \varepsilon$$

The probit model includes a single threshold value that is automatically fixed at zero for identification. The residual variance is similarly fixed at one.

The second part of the model considers only the non-zero counts. The outcome variable in the mediation model is a recoded version of Y that is missing whenever Y (or Y_{bin}) equals zero. The mediation model equations are shown below, where X is binary and M is a continuous mediator.

$$M = I_M + \alpha X + \varepsilon_M$$

$$\ln(\hat{\mu}) = I_Y + \beta M + \tau' X$$

The term inside the natural log is the predicted count given the constellation of predictors on the right side of the equation. The β and τ' coefficients reflect changes on the natural log of the count.

Geldhof et al. (2018) show that indirect effects that reflect changes to Y on the count metric are conditional on the values of X . In this example, X is binary, so there are two conditional indirect effects. They define the conditional indirect effect at a given value of X as follows

$$\alpha\beta|X = \alpha \times (\beta e^{I_Y + \beta M + \tau' X})$$

The `PARAMETERS` command in the script below computes these conditional indirect effects at each value of X .

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.6.imp](#) [data22.dat](#)

The syntax highlights are as follows.

- ❖ `ORDINAL` command identifies binary predictors and the binary outcome indicating whether the count was greater than zero
- ❖ `COUNT` command identifies a count outcome
- ❖ `TRANSFORM` command creates the variables for the two-part model
- ❖ `MODEL` command labels the indirect effect's component pathways
- ❖ `PARAMETERS` command uses labeled quantities to compute conditional indirect effects

```
DATA: data22.dat;
VARIABLES: id x v1:v4 m ycnt v5 v6;
MISSING: 999;
TRANSFORM:
y = missing(ycnt == 0, ycnt);
ybin = ifelse(ycnt == 0, 0, 1);
ORDINAL: x ybin;
COUNT: y;
MODEL:
mediation.model:
```

```

m ~ 1@m_icept x@alpha;
y ~ 1@y_icept m@beta x@tau;
binary.model:
ybin ~ x m;
PARAMETERS:
x0 = 0;
x1 = 1;
ab_at_x0 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x0));
ab_at_x1 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x1));
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding rblimp script is as follows.

```

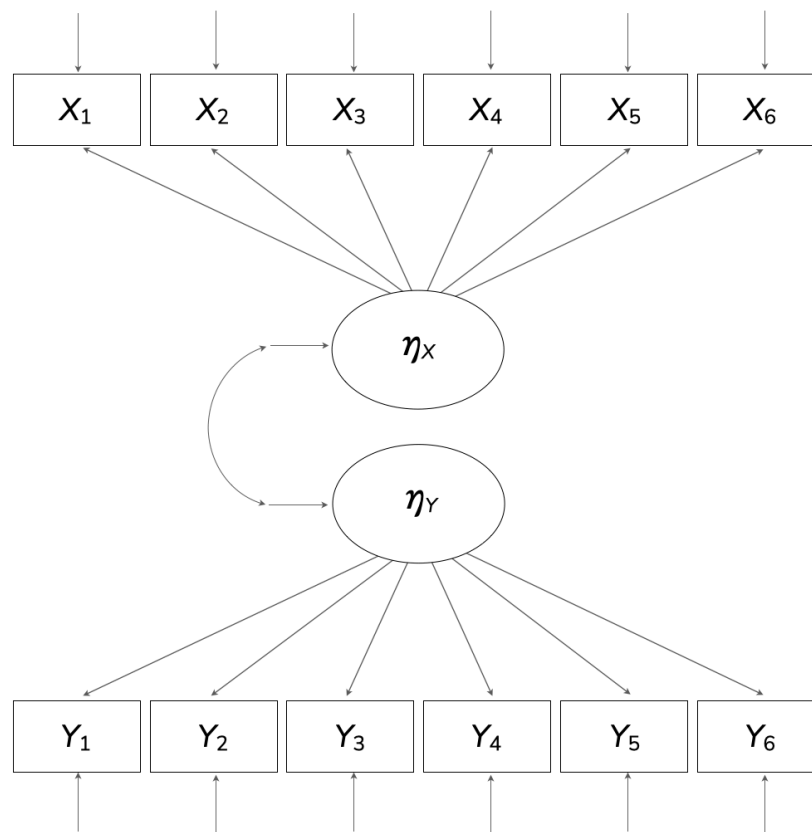
library(rblimp)
load(file = 'data22.rda')

mymodel <- rblimp(
  data = data22,
  transform = 'y = missing(ycnt == 0, ycnt);
  ybin = ifelse(ycnt == 0, 0, 1)',
  ordinal = 'x ybin',
  count = 'y',
  model = '
mediation.model:
m ~ 1@m_icept x@alpha;
y ~ 1@y_icept m@beta x@tau;
binary.model:
ybin ~ x m',
  parameters = 'x0 = 0;
x1 = 1;
ab_at_x0 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x0));
ab_at_x1 = alpha * (beta*exp(y_icept + beta*m_icept + tau*x1))',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```

5.7: CFA With Continuous Indicators

This example illustrates a two-factor measurement model with correlated latent variables, each measured by six continuous indicators. A path diagram of the analysis model is shown below.



Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.7.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ `LATENT` command defines two latent variables
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output

- ❖ **MODEL** command fixes variances and residual variances to one for identification
- ❖ **PARAMETERS** command specifies a truncated prior over positive values, and the prior is attached to each factor's first loading in the **MODEL** command

```

DATA: data4.dat;
VARIABLES: id v1:v9 y1:y6 v10:v16 x1:x6;
MISSING: 999;
LATENT: latency latentx;
MODEL:
  latent.model:
  latentx ~~ latentx@1;
  latency  ~~ latency@1;
  latentx  ~~ latency;
  measurement.models:
  latentx -> x1@xload_prior x2:x6;
  latency  -> y1@yload_prior y2:y6;
PARAMETERS:
  xload_prior ~ truncate(0, Inf);
  yload_prior ~ truncate(0, Inf);
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  latent = 'latency latentx',
  model = '
  latent.model:
  latentx ~~ latentx@1;
  latency  ~~ latency@1;
  latentx  ~~ latency;
  measurement.models:
  latentx -> x1@xload_prior x2:x6;
  latency  -> y1@yload_prior y2:y6',
  parameters = 'xload_prior ~ truncate(0,Inf);
  yload_prior ~ truncate(0,Inf)',

```

```

seed = 90291,
burn = 10000,
iter = 10000
)
output(mymodel)

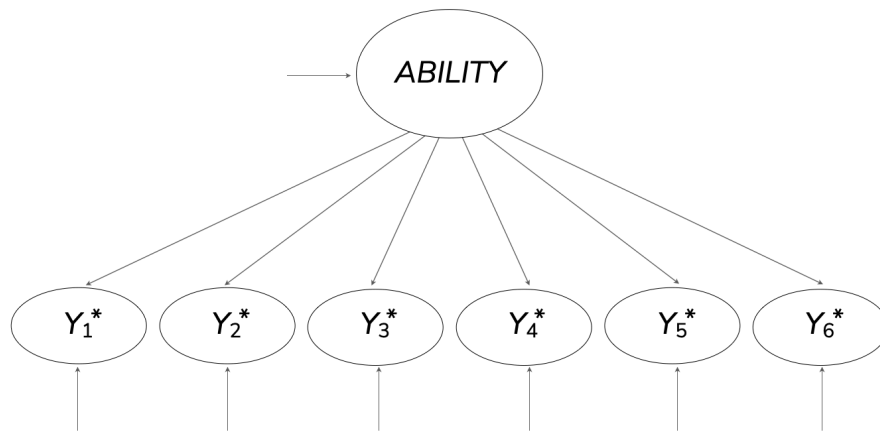
```

5.8: CFA With Binary Indicators (Two-Parameter IRT Model)

This example illustrates a unidimensional measurement model with binary indicators and IRT scaling. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.6a.imp](#) [Ex5.8b.imp](#) [data4.dat](#)

A path diagram of the analysis model is shown below, with ellipses denoting latent response variables, the residual variances of which are fixed scaling constants.



Blimp can use either a logit or probit link. The syntax highlights for the logistic link are as follows.

- ❖ **ORDINAL** command identifies binary variables
- ❖ **LATENT** command defines a latent (ability) variable
- ❖ **MODEL** command fixes the mean and variance of the latent variable to zero and one, respectively

- ❖ **MODEL** command labels measurement intercepts and factor loadings
- ❖ Applying the **logit** function to the dependent variable on the **MODEL** line requests a logit rather than probit link
- ❖ **PARAMETERS** command uses labeled quantities to compute item discrimination and difficulty indices for 2-parameter IRT scaling

```
DATA: data14.dat;
VARIABLES: id y1:y6;
ORDINAL: y1:y6;
MISSING: 999;
LATENT: ability;
MODEL:
ability ~ 1@0;
ability ~~ ability@1;
logit(y1) ~ 1@icept1 ability@load1;
logit(y2) ~ 1@icept2 ability@load2;
logit(y3) ~ 1@icept3 ability@load3;
logit(y4) ~ 1@icept4 ability@load4;
logit(y5) ~ 1@icept5 ability@load5;
logit(y6) ~ 1@icept6 ability@load6;
PARAMETERS:
discrim1 = load1;
discrim2 = load2;
discrim3 = load3;
discrim4 = load4;
discrim5 = load5;
discrim6 = load6;
difficulty1 = - icept1 / load1;
difficulty2 = - icept2 / load2;
difficulty3 = - icept3 / load3;
difficulty4 = - icept4 / load4;
difficulty5 = - icept5 / load5;
difficulty6 = - icept6 / load6;
SEED: 90291;
BURN: 2000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data14.rda')
```

```

mymodel <- rblimp(
  data = data14,
  ordinal = 'y1:y6',
  latent = 'ability',
  model = '
ability ~ 1@0;
ability ~~ ability@1;
logit(y1) ~ 1@iccept1 ability@load1;
logit(y2) ~ 1@iccept2 ability@load2;
logit(y3) ~ 1@iccept3 ability@load3;
logit(y4) ~ 1@iccept4 ability@load4;
logit(y5) ~ 1@iccept5 ability@load5;
logit(y6) ~ 1@iccept6 ability@load6',
  parameters = 'discrim1 = load1;
discrim2 = load2;
discrim3 = load3;
discrim4 = load4;
discrim5 = load5;
discrim6 = load6;
difficulty1 = - iccept1 / load1;
difficulty2 = - iccept2 / load2;
difficulty3 = - iccept3 / load3;
difficulty4 = - iccept4 / load4;
difficulty5 = - iccept5 / load5;
difficulty6 = - iccept6 / load6',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

```

The script below is identical but uses a probit rather than logit link (i.e., a normal ogive model specification). The logistic coefficients differ by a factor of approximately 1.7.

```

DATA: data14.dat;
VARIABLES: id y1:y6;
ORDINAL: y1:y6;
MISSING: 999;
LATENT: ability;
MODEL:

```

```

ability ~ 1@0;
ability ~~ ability@1;
y1 ~ 1@icept1 ability@load1;
y2 ~ 1@icept2 ability@load2;
y3 ~ 1@icept3 ability@load3;
y4 ~ 1@icept4 ability@load4;
y5 ~ 1@icept5 ability@load5;
y6 ~ 1@icept6 ability@load6;
PARAMETERS:
discrim1 = load1;
discrim2 = load2;
discrim3 = load3;
discrim4 = load4;
discrim5 = load5;
discrim6 = load6;
difficulty1 = - icept1 / load1;
difficulty2 = - icept2 / load2;
difficulty3 = - icept3 / load3;
difficulty4 = - icept4 / load4;
difficulty5 = - icept5 / load5;
difficulty6 = - icept6 / load6;
SEED: 90291;
BURN: 3000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data14.rda')

mymodel <- rblimp(
  data = data14,
  ordinal = 'y1:y6',
  latent = 'ability',
  model = '
ability ~ 1@0;
ability ~~ ability@1;
y1 ~ 1@icept1 ability@load1;
y2 ~ 1@icept2 ability@load2;
y3 ~ 1@icept3 ability@load3;
y4 ~ 1@icept4 ability@load4;
y5 ~ 1@icept5 ability@load5;
y6 ~ 1@icept6 ability@load6',

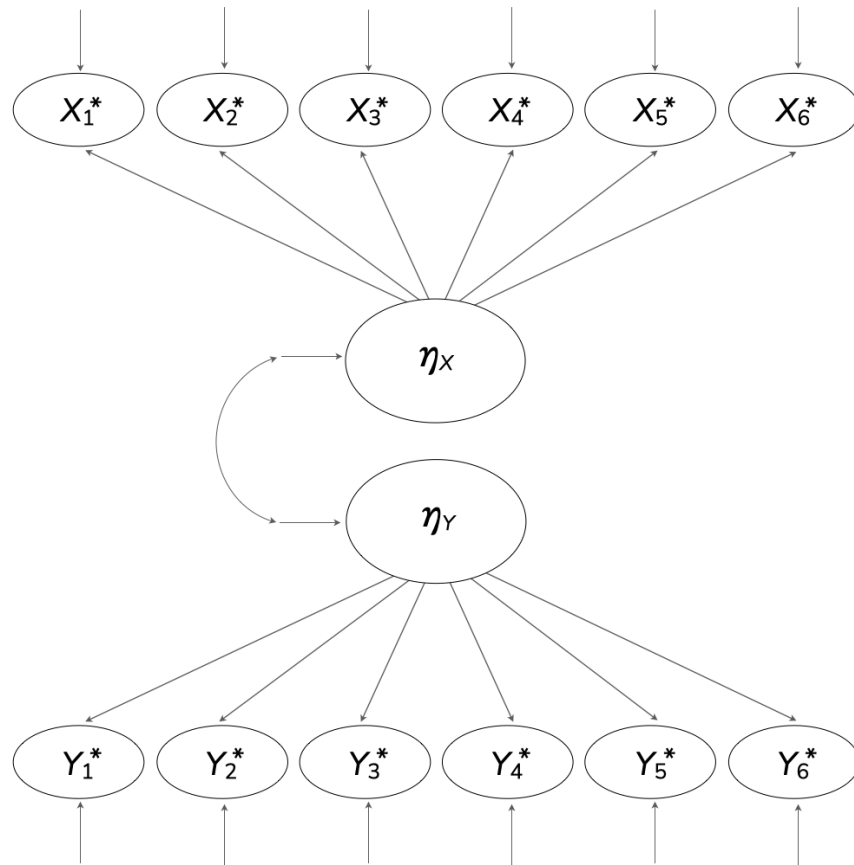
```



```
parameters = 'discrim1 = load1;
discrim2 = load2;
discrim3 = load3;
discrim4 = load4;
discrim5 = load5;
discrim6 = load6;
difficulty1 = - iclept1 / load1;
difficulty2 = - iclept2 / load2;
difficulty3 = - iclept3 / load3;
difficulty4 = - iclept4 / load4;
difficulty5 = - iclept5 / load5;
difficulty6 = - iclept6 / load6',
seed = 90291,
burn = 3000,
iter = 10000
)
output(mymodel)
```

5.9: CFA With Ordinal Indicators

This example illustrates a two-factor measurement model with correlated latent variables, each measured by six ordinal indicators. A path diagram of the analysis model is shown below, with ellipses denoting latent response variables, the residual variances of which are fixed at one.



Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.9.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies ordinal variables
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ **LATENT** command defines two latent variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command fixes variances and residual variances to one for identification

- ❖ **PARAMETERS** command specifies a truncated prior over positive values, and the prior is attached to each factor's first loading in the **MODEL** command
- ❖ Longer burn-in period for ordered categorical variables

```

DATA: data4.dat;
VARIABLES: id v1:v9 y1:y6 v10:v16 x1:x6;
ORDINAL: x1:x6 y1:y6;
MISSING: 999;
LATENT: latency latentx;
MODEL:
  latent.model:
  latentx ~~ latentx@1;
  latency  ~~ latency@1;
  latentx  ~~ latency;
  measurement.models:
  latentx  -> x1@xload_prior x2:x6;
  latency  -> y1@yload_prior y2:y6;
PARAMETERS:
  xload_prior ~ truncate(0, Inf);
  yload_prior ~ truncate(0, Inf);
SEED: 90291;
BURN: 50000;
ITER: 50000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'x1:x6 y1:y6',
  latent = 'latency latentx',
  model = '
  latent.model:
  latentx ~~ latentx@1;
  latency  ~~ latency@1;
  latentx  ~~ latency;
  measurement.model:
  latentx  -> x1@xload_prior x2:x6;
  latency  -> y1@yload_prior y2:y6',

```

```
parameters = 'xload_prior ~ truncate(0, Inf);  
yload_prior ~ truncate(0, Inf)',  
seed = 90291,  
burn = 50000,  
iter = 50000  
)  
output(mymodel)
```

5.10: Imputing Latent Response Scores for Item-Level Factor Analysis

Examples 5.8 and 5.9 illustrated item-level factor analyses that imposed a measurement model on latent response variables. This example illustrates a latent variable imputation scheme from Enders (2022) that creates multiple imputation data sets containing categorical items as well as their underlying latent response variables (i.e., plausible values). The goal is to convert a categorical factor analysis problem into a normal-theory multiple imputation analysis that uses the latent response scores as indicators in lieu of discrete items. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.10.imp](#) [Ex5.10.R](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies ordinal variables
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ **FCS** command specifies fully conditional specification multiple imputation
- ❖ **savelatent** keyword on the **OPTIONS** line saves latent response scores
- ❖ **NIMPS** command specifies 20 imputed data sets
- ❖ Longer burn-in period for ordered categorical variables
- ❖ Setting **CHAINS** equal to **NIMPS** saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```

DATA: data4.dat;
VARIABLES: id v1:v9 y1:y6 v10:v16 x1:x6;
ORDINAL: x1:x6 y1:y6;
MISSING: 999;
FCS: x1:x6 y1:y6;
SEED: 90291;
BURN: 25000;
ITER: 10000;
CHAINS: 100;
NIMPS: 100;
OPTIONS: savelatent;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed. The latent response variables have a `.latent` suffix appended to the discrete variable's name.

VARIABLE ORDER IN IMPUTED DATA:

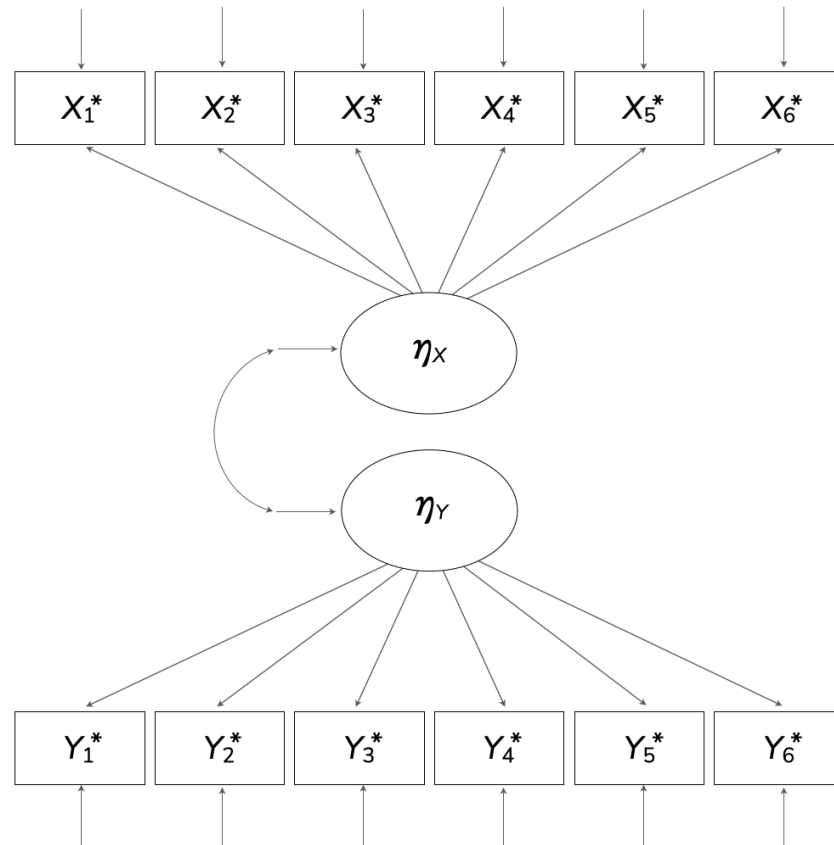
```

stacked = 'imps.dat'

imp# id v1 v2 v3 v4 v5 v6 v7 v8 v9 y1 y2 y3 y4 y5 y6 v10
v11 v12 v13 v14 v15 v16 x1 x2 x3 x4 x5 x6
y1.latent y2.latent y3.latent
y4.latent y5.latent y6.latent
x1.latent x2.latent x3.latent
x4.latent x5.latent x6.latent

```

In the analysis phase, a normal-theory item-level confirmatory factor analysis is fit to the imputed latent response scores using other software packages. A path diagram is as follows.



R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp_fcs` to create multiple imputations, and it uses `lavaan` (Rosseel, Jorgensen, & Rockwood, 2021) and `lavaan.mi` (Jorgensen, 2024) to fit a two-factor measurement model to the latent normal imputations. Note that the `MISSING` and `FCS` commands are no longer necessary.. The former is omitted because that information is contained in the R data file. The `FCS` command is replaced by a `variables` parameter that lists the variables to be included in the imputation model. Additionally, the `SAVE` command and `saveLatent` keyword on the `OPTIONS` line are no longer necessary because imputations and latent variable scores are automatically stored in an `rblimp` list object called `myModel@imputations`. The resulting estimates are numerically equivalent to applying full information maximum likelihood analysis (FIML) with a probit link to the categorical data, but the FIML analysis often doesn't provide fit indices because the saturated model is too complex to estimate.

```

library(fdir)
library(rblimp)
library(lavaan.mi)
load(file = 'data4.rda')

mymodel <- rblimp_fcs(
  data = data4,
  ordinal = 'x1:x6 y1:y6',
  variables = 'x1:x6 y1:y6',
  seed = 90291,
  burn = 25000,
  iter = 10000,
  chains = 20,
  nimps = 20
)
output(mymodel)

# inspect variable names
names(mymodel@imputations[[1]])

# mitml list
implist <- as.mitml(mymodel)

# specify cfa model with latent response imputations
lavaan_model <- c(
  paste('ylatent =~', paste0('y', 1:6, '.latent', collapse = ' + ')),
  paste('xlatent =~', paste0('x', 1:6, '.latent', collapse = ' + ')),
  'ylatent ~~ xlatent', 'ylatent ~~ 1*ylatent', 'xlatent ~~ 1*xlatent')

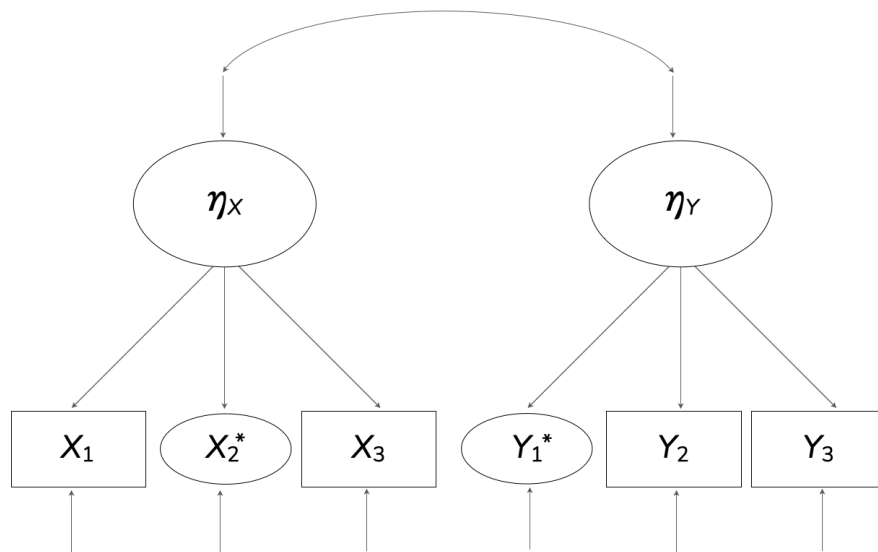
# fit model with lavaan.mi
results <- cfa.mi(lavaan_model, data = implist, estimator = "ml")
summary(results, standardized = T, fit = T)

# imputation-based modification indices
modindices.mi(results, op = c("~=", "=~"), minimum.value = 3, sort. = T)

```

5.11: Skewed Indicators With a Yeo-Johnson Transformation

This example illustrates a two-factor model with correlated latent variables, each measured by three continuous indicators. One indicator from each latent factor is skewed, and a Yeo-Johnson (Yeo & Johnson, 2000) normalizing transformation is applied to these indicators. A path diagram of the analysis model is shown below.



The ellipses indicate normalized indicators, which are essentially latent normal variables that have a nonlinear mapping to the nonnormal manifest variables. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.11.imp](#) [Ex5.11.R](#) [data12.dat](#)

The syntax highlights are as follows.

- ❖ LATENT command defines two latent variables
- ❖ Applying `yjt` function to skewed indicators on the MODEL line requests a Yeo-Johnson transformation
- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command fixes variances and residual variances to one for identification

```
DATA: data12.dat;
VARIABLES: x1:x3 y1:y3;
MISSING: 999;
LATENT: latentx latenty;
```



```

MODEL:
structural.model:
latentx ~~ latentx@1;
latency ~~ latency@1;
latentx ~~ latency;
measurement.models:
latentx -> x1 x3;
yjt(x2) ~ latentx;
yjt(y1) ~ latency;
latency -> y2 y3;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data12.rda')

mymodel <- rblimp(
  data = data12,
  latent = 'latentx latency',
  model = '
  structural.model:
  latentx ~~ latentx@1;
  latency ~~ latency@1;
  latentx ~~ latency;
  measurement.models:
  latentx -> x1 x3;
  yjt(x2) ~ latentx;
  yjt(y1) ~ latency;
  latency -> y2 y3',
  seed = 90291,
  burn = 10000,
  iter = 10000
)
output(mymodel)

```

Blimp can save multiple imputations from any model it estimates. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis, and listing the `savelatent` keyword on the `OPTIONS` command saves the normalized imputes from the Yeo-Johnson transformation alongside the skewed

imputes on the raw score metric (this keyword also saves the latent response scores for the binary predictor).

```

DATA: data12.dat;
VARIABLES: x1:x3 y1:y3;
MISSING: 999;
LATENT: latentx latenty;
MODEL:
  structural.model:
  latentx ~~ latentx@1;
  latenty ~~ latenty@1;
  latentx ~~ latenty;
  measurement.models:
  latentx -> x1 x3;
  yjt(x2) ~ latentx;
  yjt(y1) ~ latenty;
  latenty -> y2 y3;
SEED: 90291;
BURN: 10000;
ITER: 10000;
OPTIONS: savelatent;
CHAINS: 20;
NIMPS: 20;
SAVE: stacked = imps.dat;

```

In addition to producing Bayesian estimates of the factor model parameters, the previous code block saves normalized imputations for a frequentist analysis. Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed. The variables `yjt(yjt(x2))` and `yjt(yjt(y1))` are the normalized variables.

VARIABLE ORDER IN IMPUTED DATA:

```

stacked = 'imps.dat'

  imp# x1 x2 x3 y1 y2 y3 latentx.latent latenty.latent
  yjt(yjt(x2)) yjt(yjt(y1))

```

In the analysis phase, a normal-theory item-level confirmatory factor analysis is fit to the original and normalized variables using other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp_fcs` to create multiple imputations, and it uses `lavaan` (Rosseel, Jorgensen, & Rockwood, 2021) and `lavaan.mi` (Jorgensen, 2024) to fit a two-factor measurement model to the latent normal imputations. Note that the `MISSING` and `FCS` commands are no longer necessary. The former is omitted because that information is contained in the R data file. The `FCS` command is replaced by a `variables` parameter that lists the variables to be included in the imputation model. Additionally, the `SAVE` command and `saveLatent` keyword on the `OPTIONS` line are no longer necessary because imputations and latent variable scores are automatically stored in an `rblimp` list object called `myModel@imputations`. The resulting estimates are numerically equivalent to applying full information maximum likelihood analysis (FIML) with a probit link to the categorical data, but the FIML analysis often doesn't provide fit indices because the saturated model is too complex to estimate.

```
library(fdir)
library(rblimp)
library(lavaan.mi)
load(file = 'data4.rda')

myModel <- rblimp_fcs(
  data = data12,
  latent = 'latentx latency',
  model = '
  structural.model:
  latentx ~~ latentx@1;
  latency  ~~ latency@1;
  latentx  ~~ latency;
  measurement.models:
  latentx  -> x1 x3;
  yjt(x2) ~ latentx;
  yjt(y1) ~ latency;
  latency  -> y2 y3',
  seed = 90291,
  burn = 10000,
```

```

    iter = 10000,
    chains = 20,
    nimps = 20
  )
output(mymodel)

# inspect variable names
names(mymodel@imputations[[1]])

# mitml list
implist <- as.mitml(mymodel)

# plot raw and transformed scores
dat2plot <- do.call(rbind, implist)
hist(dat2plot$y1,breaks = 20)
hist(dat2plot$yjt.yjt.y1..,breaks = 20)

# specify cfa model with latent response imputations
lavaan_model <- c('ylatent =~ x1 + yjt.yjt.x2.. + x3',
                 'xlatent =~ yjt.yjt.y1.. + y2 + y3',
                 'ylatent ~~ xlatent',
                 'ylatent ~~ 1*ylatent','xlatent ~~ 1*xlatent')

# fit model with lavaan.mi
results <- cfa.mi(lavaan_model, data = implist, estimator = "ml")
summary(results, standardized = T, fit = T)

# imputation-based modification indices
modindices.mi(results, op = c("~~","=~"), minimum.value = 3, sort. = T)

```

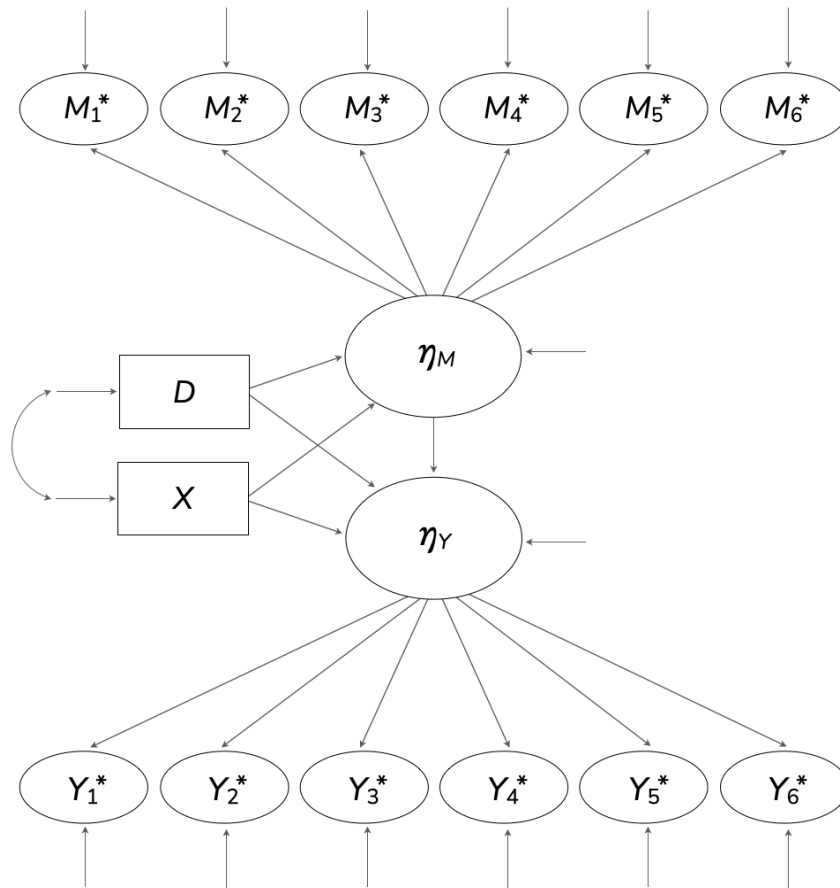
5.12: Latent Variable Regression Model

This example illustrates a latent variable mediation model where both the mediator and outcome are latent variables, each with six ordinal indicators. The structural regression equations are as follows

$$\eta_M = \beta_{01} + \beta_{11}X + \beta_{21}D + \varepsilon_1$$

$$\eta_Y = \beta_{02} + \beta_{12}\eta_M + \beta_{22}X + \beta_{32}D + \varepsilon_2$$

and a path diagram for the full model is shown below.



The residual variances of all latent response variances are fixed at values of one, and mediated pathways can be computed following Example 5.1. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.12.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies binary and ordinal variables
- ❖ **FIXED** command defines a complete predictor
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ **LATENT** command defines two latent variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output

- ❖ MODEL command fixes variances and residual variances to one for identification
- ❖ Longer burn-in period for estimating latent variables and threshold parameters

```

DATA: data4.dat;
VARIABLES: id v1:v5 x v6 v7 d y1:y6 m1:m7 v8:v13;
ORDINAL: d y1:y6 m1:m7;
MISSING: 999;
FIXED: d;
LATENT: latency latentm;
MODEL:
  structural.model:
  latentm ~ x d;
  latency ~ latentm x d;
  latentm ~~ latentm@1;
  latency ~~ latency@1;
  measurement.models:
  latentm -> m1@xload_prior m2:m6;
  latency -> y1@yload_prior y2:y6;
PARAMETERS:
  xload_prior ~ truncate(0, Inf);
  yload_prior ~ truncate(0, Inf);
SEED: 90291;
BURN: 50000;
ITER: 50000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'd y1:y6 m1:m7',
  fixed = 'd',
  latent = 'latency latentm',
  model = '
  structural.model:
  latentm ~ x d;
  latency ~ latentm x d;
  latentm ~~ latentm@1;
  latency ~~ latency@1;

```

```

measurement.models:
latentm -> m1@xload_prior m2:m6;
latenty -> y1@yload_prior y2:y6',
parameters = 'xload_prior ~ truncate(0, Inf);
yload_prior ~ truncate(0, Inf)',
seed = 90291,
burn = 50000,
iter = 50000
)
output(mymodel)

```

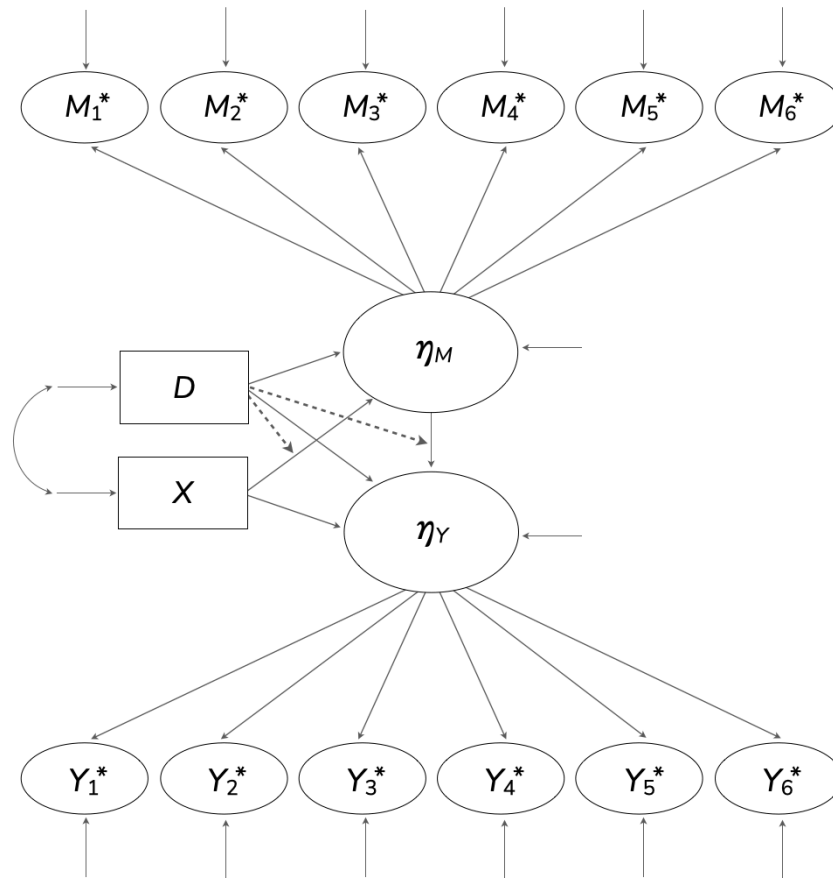
5.13: Latent-by-Manifest Variable Interaction

This example adds moderated paths to the latent variable mediation model from the previous example. The structural regression equations feature an interaction between two manifest variables and an interaction between a manifest and latent variable.

$$\eta_M = \beta_{01} + \beta_{11}X + \beta_{21}D + \beta_{31}(X)(D) + \varepsilon_1$$

$$\eta_Y = \beta_0 + \beta_1\eta_X + \beta_2D + \beta_3(\eta_X)(D) + \varepsilon_2$$

The path diagram of the full model is shown below.



The dashed lines pointing from D to the directed arrows convey that D moderates the association between X and the latent mediator as well as the association between the latent mediator and the outcome. The residual variances of all latent response variances are fixed at values of one, and mediated pathways can be computed following Example 5.2. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.13.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies binary and ordinal variables
- ❖ **FIXED** command defines a complete predictor

- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ LATENT command defines two latent variables
- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command fixes variances and residual variances to one for identification
- ❖ MODEL command features product terms
- ❖ Longer burn-in period for ordered categorical variables

```

DATA: data4.dat;
VARIABLES: id v1:v5 x v6 v7 d y1:y6 m1:m7 v8:v13;
ORDINAL: d y1:y6 m1:m7;
MISSING: 999;
FIXED: d;
LATENT: latency latentm;
MODEL:
  structural.model:
  latentm ~ x d x*d;
  latency ~ latentm x d latentm*d;
  latentm ~~ latentm@1;
  latency ~~ latency@1;
  measurement.models:
  latentm -> m1@xload_prior m2:m6;
  latency -> y1@yload_prior y2:y6;
PARAMETERS:
  xload_prior ~ truncate(0, Inf);
  yload_prior ~ truncate(0, Inf);
SEED: 90291;
BURN: 50000;
ITER: 50000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'd y1:y6 m1:m7',
  fixed = 'd',

```

```

latent = 'latent latentm',
model = '
structural.model:
latentm ~ x d x*d;
latent ~ latentm x d latentm*d;
latentm ~~ latentm@1;
latent ~~ latent@1;
measurement.models:
latentm -> m1@xload_prior m2:m6;
latent -> y1@yload_prior y2:y6',
parameters = 'xload_prior ~ truncate(0, Inf);
yload_prior ~ truncate(0, Inf)',
seed = 90291,
burn = 50000,
iter = 50000
)
output(mymodel)

```

5.14: Moderated Nonlinear Factor Analysis (MNLFA)

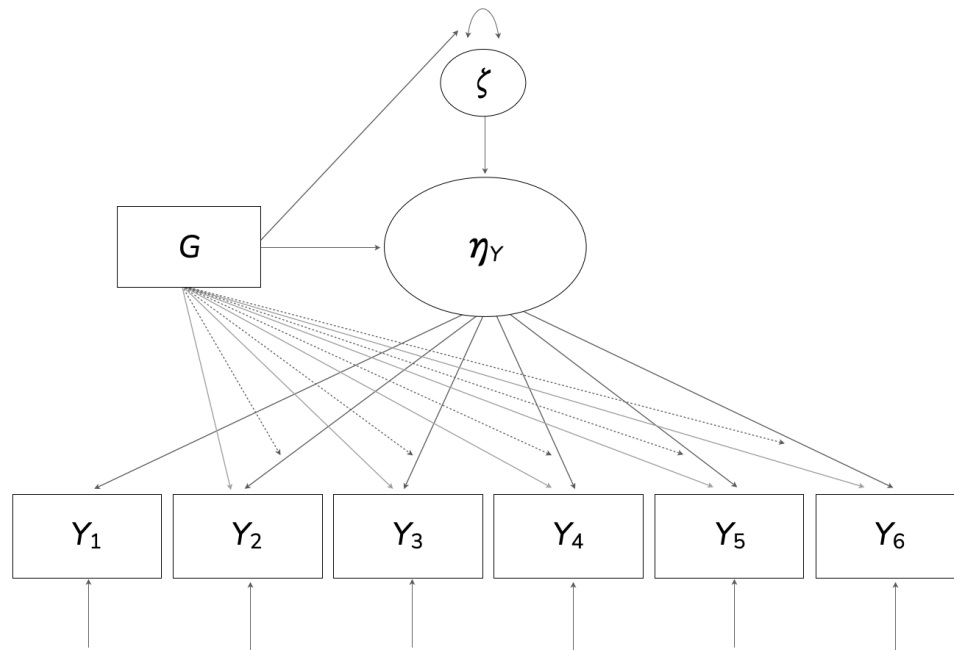
This example illustrates a moderated factor analysis for measurement invariance testing. The method is conceptually similar to nonlinear moderated factor analysis (Bauer, 2017) but does not require complicated constraints. Instead, the model represents differential loadings as an interaction between a latent variable and a manifest variable. In this example, the manifest variable is a grouping variable, but it could be any metric that Blimp supports. The latent variable's variance and the indicator variances can also be modeled as a function of the background variable, which does not need to be categorical. Enders, Vera, Keller, Lenartowicz, Loo (2024) describe the model in detail and provide a real-data analysis example involving an integrative data analysis. The data analysis scripts for this more thorough illustration are available at <https://osf.io/gfu5m/>.

The measurement model equation for a manifest indicator k is as follows.

$$Y_k = \nu_k + \lambda_k \eta_Y + \gamma_{1k} G + \gamma_{2k}(G)(\eta_Y) + \varepsilon_k$$

The γ_1 slope is the effect of the covariate on the indicator's mean, and the γ_2 interaction coefficient captures loading changes as a function of G . In this example,

the indicators are continuous, but they could be any metric that Blimp supports. A path diagram of the model is shown below.



The straight lines from G to the indicators introduce group differences in measurement intercepts, and the dashed lines from G to the directed arrows reflect manifest-by-latent interaction terms (factor loading differences). Unlike a conventional multiple-group model, G could be a continuous dimension, although it is binary in this example. Finally, the model includes a log linear equation that relates the variance of the latent factor to the background variable.

$$\ln(\sigma_{\zeta}^2) = \omega_0 + \omega_1 G$$

For identification, the intercept ω_0 is fixed at zero on the logarithmic metric, which sets the factor variance of the $G = 0$ group to one.

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.14.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies a binary predictor
- ❖ **FIXED** command identifies a complete predictor
- ❖ **LATENT** command defines a latent variable
- ❖ Individual regression equations specified for each indicator (instead of the -> convention for latent factors)
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command labels intercept group differences and loading group differences
- ❖ **MODEL** command features product terms
- ❖ **MODEL** command uses an @ and a label to assign a positive-valued prior distribution to the first factor loading
- ❖ **MODEL** command features a regression equation with the natural logarithm of the factor variance as the outcome
- ❖ **PARAMETER** command defines a positive-valued uniform prior distribution
- ❖ **WALDTEST** commands specify Bayesian Wald tests that evaluate the null hypothesis that intercept and loading differences equal 0
- ❖ **SIMPLE** command produces conditional effects (group-specific intercepts and loadings) at each level of the binary moderator
- ❖ Longer burn-in period for estimating latent variables

```
DATA: data4.dat;  
VARIABLES: id v1:v8 g y1:y6 v9:v21;  
ORDINAL: g;  
MISSING: 999;  
FIXED: g;  
LATENT: latency;
```

```

MODEL:
structural.model:
latency ~ 1@0 g;
var(latency) ~ 1@0 g;
measurement.model:
y1 ~ 1 latency@load_prior;
y2 ~ g@diffcept1 latency g*latency@diffload1;
y3 ~ g@diffcept2 latency g*latency@diffload2;
y4 ~ g@diffcept3 latency g*latency@diffload3;
y5 ~ g@diffcept4 latency g*latency@diffload4;
y6 ~ g@diffcept5 latency g*latency@diffload5;
WALDTEST: diffload1:diffload5 = 0;
WALDTEST: diffcept1:diffcept5 = 0;
PARAMETERS:
load_prior ~ truncate(0, Inf);
SIMPLE: latency | g;
SEED: 90291;
BURN: 20000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  ordinal = 'g',
  fixed = 'g',
  latent = 'latency',
  model = '
structural.model:
latency ~ 1@0 g;
var(latency) ~ 1@0 g;
measurement.model:
y1 ~ 1 latency@load_prior;
y2 ~ g@diffcept1 latency g*latency@diffload1;
y3 ~ g@diffcept2 latency g*latency@diffload2;
y4 ~ g@diffcept3 latency g*latency@diffload3;
y5 ~ g@diffcept4 latency g*latency@diffload4;
y6 ~ g@diffcept5 latency g*latency@diffload5',
  waldtest = list('diffload1:diffload5 = 0',
    'diffcept1:diffcept5 = 0')

```

```

parameters = 'load_prior ~ truncate(0, Inf)',
simple = 'latency | g',
seed = 90291,
burn = 20000,
iter = 10000
)
output(mymodel)

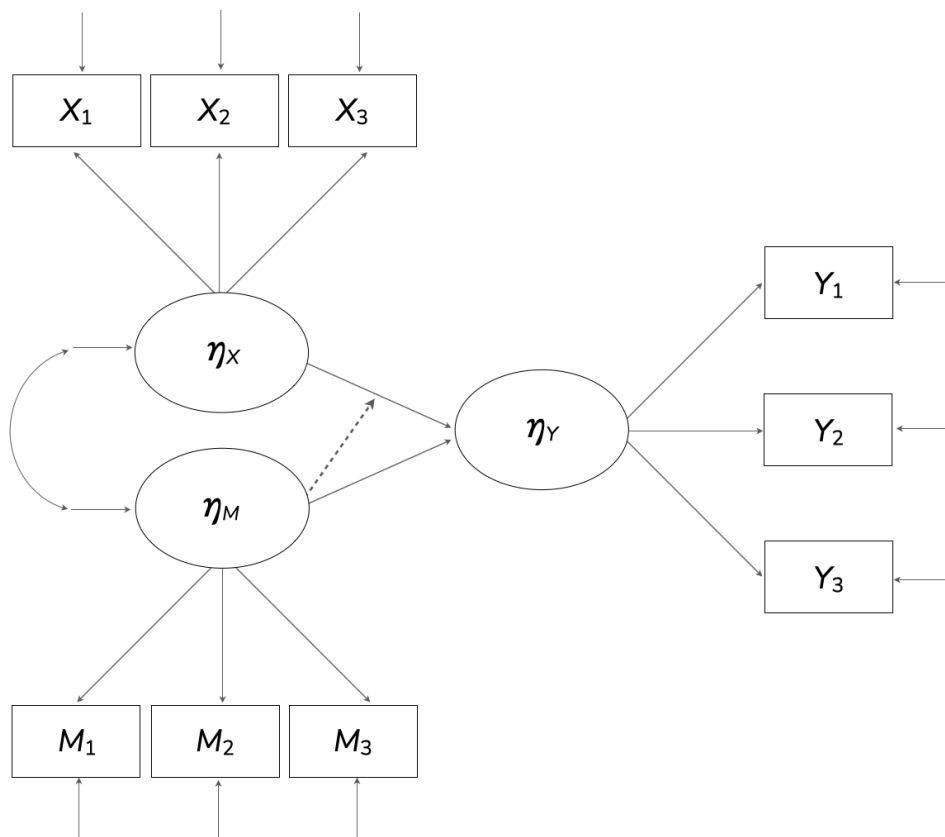
```

5.15: Latent-by-Latent Variable Interaction

This example illustrates a latent variable regression model with two latent predictors and their interaction influencing a latent outcome variable. The structural regression equation is as follows.

$$\eta_Y = \beta_0 + \beta_1\eta_X + \beta_2\eta_M + \beta_3(\eta_X)(\eta_M) + \varepsilon$$

A path diagram of the full model is shown below.



The dashed line pointing from the latent variable to the directed arrow conveys that one latent predictor is moderating the influence of the other. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.15.imp](#) [data13.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies ordinal variables
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ **LATENT** command defines three latent variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command features a two-way latent product term
- ❖ **MODEL** command labels the structural regression slopes
- ❖ **MODEL** command fixes variances and residual variances to one for identification
- ❖ **PARAMETERS** command specifies a truncated prior over positive values, and the prior is attached to each factor's first loading in the **MODEL** command
- ❖ **PARAMETERS** command uses labeled quantities to compute conditional effects (simple slopes) at plus and minus one standard deviation above the latent moderator's mean

```

DATA: data13.dat;
VARIABLES: x1:x3 m1:m3 y1:y3;
MISSING: 999;
LATENT: latentx latentm latenty;
MODEL:
  structural.model:
  latenty ~ latentx@b1 latentm@b2 latentx*latentm@b3;
  latenty ~~ latenty@1;
  predictor.model:
  latentx ~~ latentx@1;
  latentm ~~ latentm@1;
  latentx ~~ latentm;
  measurement.models:
  latentx -> x1@xload_prior x2:x3;

```

```

latentm -> m1@mload_prior m2:m3;
latenty -> y1@yload_prior y2:y3;
PARAMETERS:
xload_prior ~ truncate(0, Inf);
mload_prior ~ truncate(0, Inf);
yload_prior ~ truncate(0, Inf);
xslope_mlow = b1 - b3 * (-1);
xslope_mmean = b1 * (0);
xslope_mhigh = b1 + b3 * (-1);
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'data13.rda')

mymodel <- rblimp(
  data = data13,
  latent = 'latentx latentm latenty',
  model = '
structural.model:
latenty ~ latentx@b1 latentm@b2 latentx*latentm@b3;
latenty ~~ latenty@1;
predictor.model:
latentx ~~ latentx@1;
latentm ~~ latentm@1;
latentx ~~ latentm;
measurement.models:
latentx -> x1@xload_prior x2:x3;
latentm -> m1@mload_prior m2:m3;
latenty -> y1@yload_prior y2:y3',
  parameters = 'xload_prior ~ truncate(0, Inf);
mload_prior ~ truncate(0, Inf);
yload_prior ~ truncate(0, Inf);
xslope_mlow = b1 - b3 * (-1);
xslope_mmean = b1 * (0);
xslope_mhigh = b1 + b3 * (-1)',
  seed = 90291,
  burn = 10000,
  iter = 10000
)

```



```
output(mymodel)
```

5.16: Three-Way Latent Variable Interaction

This example illustrates a latent variable regression model with three latent predictors and all possible interactions among the latent variables. The procedure for the example is described in Keller (2024). A preprint of the paper can be downloaded [here](#). The structural model below corresponds to Equation 28 from the paper. We use established generic notation in lieu of the original variable names.

$$\eta_Y = \beta_0 + \beta_1\eta_X + \beta_2\eta_Z + \beta_3\eta_M + \beta_4(\eta_X)(\eta_Z) + \beta_5(\eta_X)(\eta_M) + \beta_6(\eta_Z)(\eta_M) + \beta_7(\eta_Z)(\eta_X)(\eta_M) + \varepsilon$$

The latent variables have 10 indicators each, and the indicators use 5-point rating scales. Following Example 5.9, the measurement models feature latent response variables regressed on the latent factors (probit regression).

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.16.imp](#) [keller2024.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies ordinal variables
- ❖ Automatic threshold specification for binary and ordinal variables
- ❖ **LATENT** command defines four latent variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command features two-way product terms and a three-way product
- ❖ **MODEL** command labels the structural regression slopes
- ❖ **MODEL** command fixes variances and residual variances to one for identification
- ❖ **PARAMETERS** command specifies a truncated prior over positive values, and the prior is attached to each factor's first loading in the **MODEL** command

- ❖ **PARAMETERS** command uses labeled quantities to compute conditional two-way effects (simple interactions) at plus and minus one standard deviation above the latent moderator's mean
- ❖ Longer burn-in period and post-burn in iterations with ordinal variables

```

DATA: keller2024.dat;
VARIABLES: x1:x10 m1:m10 z1:z10 y1:y23 d1 d2;
ORDINAL: x1:x10 z1:z10 m1:m10 y1:y23;
LATENT: latentx latentm latentz latenty;
MODEL:
structural.model:
latentx ~ latentx@b1 latentz@b2 latentm@b3
  latentx*latentz@b4 latentx*latentm@b5 latentz*latentm@b6
  latentx*latentz*latentm@b7;
latenty ~ latenty@1;
predictor.model:
latentx latentz latentm ~ latentx latentz latentm;
latentx ~ latentx@1;
latentz ~ latentz@1;
latentm ~ latentm@1;
measurement.models:
latentx -> x1@xload_prior x2:x10;
latentz -> z1@zload_prior z2:z10;
latentm -> m1@mload_prior m2:m10;
latenty -> y1@yload_prior y2:y10;
PARAMETERS:
xload_prior ~ truncate(0, Inf);
zload_prior ~ truncate(0, Inf);
mload_prior ~ truncate(0, Inf);
yload_prior ~ truncate(0, Inf);
xbyz_mlow = b4 + b7 * (-1);
xbyz_mmean = b4 + b7 * ( 0);
xbyz_mhigh = b4 + b7 * (+1);
SEED: 90291;
BURN: 20000;
ITER: 50000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'keller2024.rda')

```

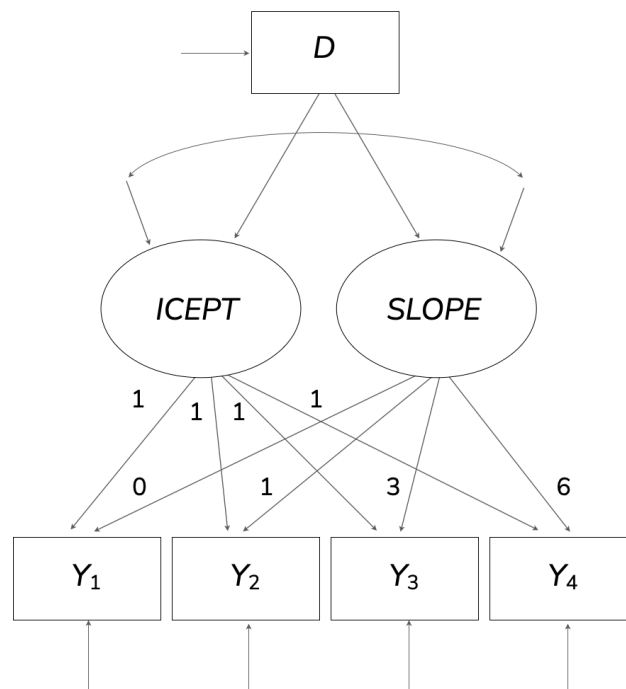
```

mymodel <- rblimp(
  data = keller2024,
  ordinal = 'x1:x10 z1:z10 m1:m10 y1:y23',
  latent = 'latentx latentm latentz latency',
  model = '
y.model:
latentx ~ latentx@b1 latentz@b2 latentm@b3
  latentx*latentz@b4 latentx*latentm@b5 latentz*latentm@b6
  latentx*latentz*latentm@b7;
latency ~ latentx@1;
predictor.model:
latentx latentz latentm ~ latentx latentz latentm;
latentx ~ latentx@1;
latentz ~ latentz@1;
latentm ~ latentm@1;
measurement.models:
latentx -> x1@xload_prior x2:x10;
latentz -> z1@zload_prior z2:z10;
latentm -> m1@mload_prior m2:m10;
latency -> y1@yload_prior y2:y10',
  parameters = 'xload_prior ~ truncate(0, Inf);
zload_prior ~ truncate(0, Inf);
mload_prior ~ truncate(0, Inf);
yload_prior ~ truncate(0, Inf);
xbyz_mlow = b4 + b7 * (-1);
xbyz_mmean = b4 + b7 * ( 0);
xbyz_mhigh = b4 + b7 * (+1)',
  seed = 90291,
  burn = 20000,
  iter = 50000
)
output(mymodel)

```

5.17: Latent Growth Curve Model

This example illustrates a two-factor latent growth curve model with unequally spaced repeated measurements and a binary predictor of the random intercepts and slopes. A path diagram of the model is shown below.



Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.17.imp](#) [data3.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies a binary predictor
- ❖ **FIXED** command identifies complete predictors
- ❖ **LATENT** command defines two latent variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ Individual regression equations specified for each indicator (instead of the -> convention for latent factors)
- ❖ **MODEL** command estimates the latent variable means, fixes the intercept factor loadings to one, fixes the growth factor loadings to the time scores (0, 1, 3, and 6), and fixes the measurement intercepts to zero
- ❖ **MODEL** command uses a label to impose equality constraint on residual variance

```

DATA: data3.dat;
VARIABLES: id y0 y1 y3 y6 d v1:v5;
ORDINAL: d;
MISSING: 999;
FIXED: d;
LATENT: icept slope;
MODEL:
  structural.model:
  icept ~ 1 d;
  slope ~ 1 d;
  icept ~~ slope;
  measurement.model:
  y0 ~ 1@0 icept@1 slope@0;
  y1 ~ 1@0 icept@1 slope@1;
  y3 ~ 1@0 icept@1 slope@3;
  y6 ~ 1@0 icept@1 slope@6;
  # common residual variance
  y0 ~~ y0@resvar;
  y1 ~~ y1@resvar;
  y3 ~~ y3@resvar;
  y6 ~~ y6@resvar;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data3.rda')

mymodel <- rblimp(
  data = data3,
  ordinal = 'd',
  fixed = 'd',
  latent = 'icept slope',
  model = '
  structural.model:
  icept ~ 1 d;
  slope ~ 1 d;
  icept ~~ slope;
  measurement.model:

```

```

y0 ~ 1@0 icept@1 slope@0;
y1 ~ 1@0 icept@1 slope@1;
y3 ~ 1@0 icept@1 slope@3;
y6 ~ 1@0 icept@1 slope@6;
y0 ~~ y0@resvar;
y1 ~~ y1@resvar;
y3 ~~ y3@resvar;
y6 ~~ y6@resvar',
seed = 90291,
burn = 10000,
iter = 10000
)
output(mymodel)

```

5.18: Residual SEM: AR1 Growth Model

This example illustrates a residual SEM that specifies an AR1 structure on the residuals in a latent growth curve model. The model features the residuals at each time point predicting the outcome at the next measurement occasion. The path diagram below shows a growth model with six waves of data. The diagram uses an asterisk to denote time-specific residuals computed by subtracting the predicted value from the observed value.

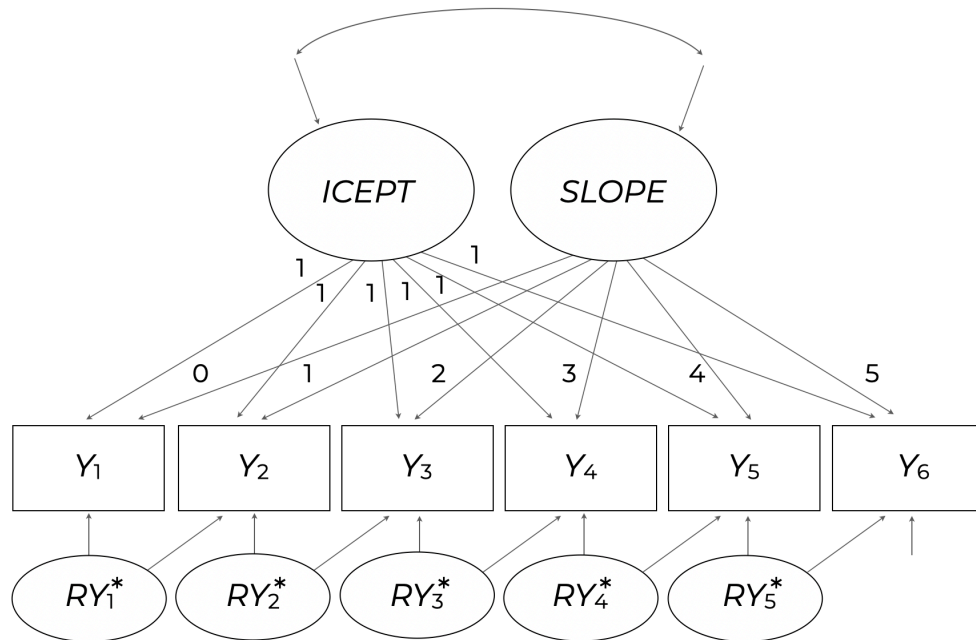
$$RY_{1i}^* = Y_{1i} - (ICEPT_i + 0 \times SLOPE_i)$$

$$RY_{2i}^* = Y_{2i} - (ICEPT_i + 1 \times SLOPE_i)$$

...

$$RY_{5i}^* = Y_{5i} - (ICEPT_i + 4 \times SLOPE_i)$$

Note that the residuals with an asterisk are not themselves random variables. Rather, they are simply functions defined by centering each repeated measures variable at its predicted value using the estimated random intercepts and slopes.



Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.18a.imp](#) [Ex5.18b.imp](#) [data27.dat](#)

The syntax highlights are as follows.

- ❖ **LATENT** command defines two latent variables that represent person-specific random intercepts and slopes
- ❖ **MODEL** command defines the computations that create the residuals, and it uses those aliases as predictors in the measurement models
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command uses the `->` convention to add random slopes as a predictor to each measurement model
- ❖ **MODEL** command estimates the latent variable means, fixes the intercept factor loadings to one, fixes the growth factor loadings to integer values between zero and six

- ❖ **WALDTEST** command invokes a Bayesian Wald test that evaluates the null hypothesis that the autocorrelations are equal

```

DATA: data27.dat;
VARIABLES: y1:y6 z;
MISSING: 999;
LATENT: icept slope;
MODEL:
# define residuals for embedded functions in measurement models
ry1 = y1 - (icept + (0*slope));
ry2 = y2 - (icept + (1*slope));
ry3 = y3 - (icept + (2*slope));
ry4 = y4 - (icept + (3*slope));
ry5 = y5 - (icept + (4*slope));
structural.model:
icept ~ slope;
1 -> icept slope;
measurement.model:
{ y1:y6 } ~ 1@icept;
slope -> y1@0 y2@1 y3@2 y4@3 y5@4 y6@5;
# autocorrelations
y2 ~ ry1@ac1;
y3 ~ ry2@ac2;
y4 ~ ry3@ac3;
y5 ~ ry4@ac4;
y6 ~ ry5@ac5;
WALDTEST: ac1 = ac2:ac5;
SEED: 90291;
BURN: 30000;
ITER: 20000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data27.rda')

mymodel <- rblimp(
  data = data27,
  latent = 'icept slope',
  model = '
ry1 = y1 - (icept + (0*slope));
ry2 = y2 - (icept + (1*slope));

```



```

ry3 = y3 - (icept + (2*slope));
ry4 = y4 - (icept + (3*slope));
ry5 = y5 - (icept + (4*slope));
structural.model:
icept ~ slope;
1 -> icept slope;
measurement.model:
{ y1:y6 } ~ 1@icept;
slope -> y1@0 y2@1 y3@2 y4@3 y5@4 y6@5;
y2 ~ ry1@ac1;
y3 ~ ry2@ac2;
y4 ~ ry3@ac3;
y5 ~ ry4@ac4;
y6 ~ ry5@ac5',
waldtest = 'ac1 = ac2:ac5',
seed = 90291,
burn = 30000,
iter = 20000
)
output(mymodel)

```

The MCMC Wald test was not significant, indicating that a simpler model where all autocorrelation paths are equal is preferable. The code block below illustrates equality constraints on the residual associations.

```

DATA: data27.dat;
VARIABLES: y1:y6 z;
MISSING: 999;
LATENT: icept slope;
MODEL:
# define residuals for embedded functions in measurement models
ry1 = y1 - (icept + (0*slope));
ry2 = y2 - (icept + (1*slope));
ry3 = y3 - (icept + (2*slope));
ry4 = y4 - (icept + (3*slope));
ry5 = y5 - (icept + (4*slope));
structural.model:
icept ~ slope;
1 -> icept slope;
measurement.model:
{ y1:y6 } ~ 1@icept;
slope -> y1@0 y2@1 y3@2 y4@3 y5@4 y6@5;

```

```
# autocorrelations
y2 ~ ry1@ac;
y3 ~ ry2@ac;
y4 ~ ry3@ac;
y5 ~ ry4@ac;
y6 ~ ry5@ac;
SEED: 90291;
BURN: 20000;
ITER: 20000;
```

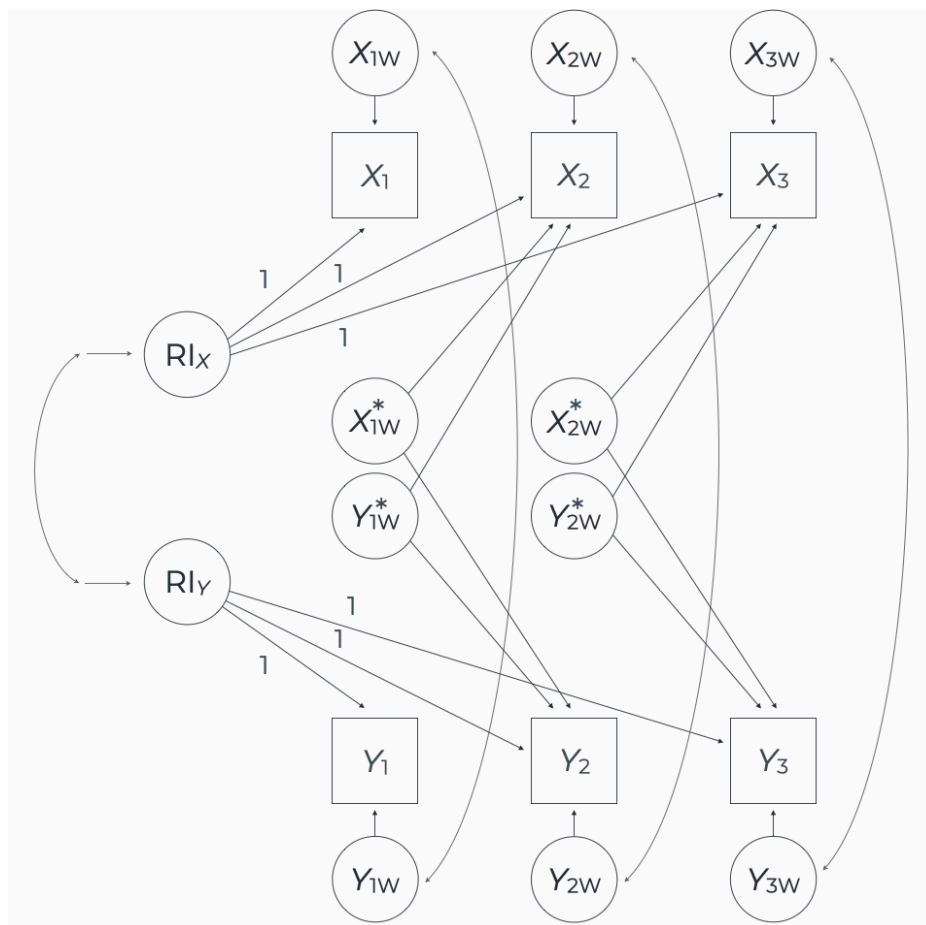
The corresponding rblimp script is as follows.

```
library(rblimp)
load(file = 'data27.rda')

mymodel <- rblimp(
  data = data27,
  latent = 'icept slope',
  model = '
ry1 = y1 - (icept + (0*slope));
ry2 = y2 - (icept + (1*slope));
ry3 = y3 - (icept + (2*slope));
ry4 = y4 - (icept + (3*slope));
ry5 = y5 - (icept + (4*slope));
structural.model:
icept ~~ slope;
1 -> icept slope;
measurement.model:
{ y1:y6 } ~ 1@icept;
slope -> y1@0 y2@1 y3@2 y4@3 y5@4 y6@5;
y2 ~ ry1@ac;
y3 ~ ry2@ac;
y4 ~ ry3@ac;
y5 ~ ry4@ac;
y6 ~ ry5@ac',
  seed = 90291,
  burn = 20000,
  iter = 20000
)
output(mymodel)
```

5.19: Random Intercept Cross-Lagged Panel Model (RI-CLPM)

This example illustrates a random intercept cross-lagged panel model. The analysis and the data are from Mulder and Hamaker (2021). The simulated data were obtained from [the paper's website](#). The basic RI-CLPM is described in Hamaker, Kuipers, and Grasman (2015). The model features a random intercept latent variable that removes stable individual differences from the lagged processes, which represent pure within-person effects. The path diagram below shows a three-wave version of the model.



The W subscripts denote within-person effects and sources of variation. The cross-lagged paths are defined as within-person slopes because the manifest variables are centered at the random intercepts, thus removing stable

between-person variation. Specifically, the residuals denoted with an asterisk in the diagram are computed as follows.

$$X_{1W}^* = X_1 - RI_X \quad X_{2W}^* = X_2 - RI_X$$

$$Y_{1W}^* = Y_1 - RI_Y \quad Y_{2W}^* = Y_2 - RI_Y$$

Note that the residuals with an asterisk are not themselves random variables. Rather, they are simply functions defined by centering each repeated measures variable at the estimated random intercept. Finally, the residual variances and covariances are within-person effects by virtue of the random intercept latent variable removing stable between-person variation.

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex5.19a.imp](#) [Ex5.19b.imp](#) [Ex5.19c.imp](#) [Ex5.19d.imp](#) [Ex5.19d.imp](#) [RICLPM.dat](#)

The syntax highlights are as follows.

- ❖ `LATENT` command defines two latent variables that represent person-specific random intercepts for the two variables
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ `MODEL` command leverages Blimp's default, which is to set latent variable means equal to zero
- ❖ `MODEL` command sets the random intercept latent variable's factor loadings to one
- ❖ `MODEL` command creates pure within-person predictors by centering each person's scores at their random intercept estimates
- ❖ `MODEL` command defines the computations for the centering procedure that creates the residuals, and it uses those aliases as predictors in the regressions

```

DATA: RICLPM.dat;
VARIABLES: x1:x5 y1:y5 z2 z1;
LATENT: RIx RIy;
MODEL:
x1r = x1 - (mux1 + RIx);
x2r = x2 - (mux2 + RIx);
x3r = x3 - (mux3 + RIx);
x4r = x4 - (mux4 + RIx);
y1r = y1 - (muy1 + RIy);
y2r = y2 - (muy2 + RIy);
y3r = y3 - (muy3 + RIy);
y4r = y4 - (muy4 + RIy);
random.intercepts:
RIx ~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1;
x2 ~ 1@mux2 RIx@1 x1r y1r;
x3 ~ 1@mux3 RIx@1 x2r y2r;
x4 ~ 1@mux4 RIx@1 x3r y3r;
x5 ~ 1@mux5 RIx@1 x4r y4r;
y.models:
y1 ~ 1@muy1 RIy@1;
y2 ~ 1@muy2 RIy@1 y1r x1r;
y3 ~ 1@muy3 RIy@1 y2r x2r;
y4 ~ 1@muy4 RIy@1 y3r x3r;
y5 ~ 1@muy5 RIy@1 y4r x4r;
covariances:
x1 ~ y1;
x2 ~ y2;
x3 ~ y3;
x4 ~ y4;
x5 ~ y5;
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'RICLPM.rda')

mymodel <- rblimp(
  data = RICLPM,

```

```

latent = 'RIx RIy',
model = '
x1r = x1 - (mux1 + RIx);
x2r = x2 - (mux2 + RIx);
x3r = x3 - (mux3 + RIx);
x4r = x4 - (mux4 + RIx);
y1r = y1 - (muy1 + RIy);
y2r = y2 - (muy2 + RIy);
y3r = y3 - (muy3 + RIy);
y4r = y4 - (muy4 + RIy);
random.intercept:
RIx ~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1;
x2 ~ 1@mux2 RIx@1 x1r y1r;
x3 ~ 1@mux3 RIx@1 x2r y2r;
x4 ~ 1@mux4 RIx@1 x3r y3r;
x5 ~ 1@mux5 RIx@1 x4r y4r;
y.models:
y1 ~ 1@muy1 RIy@1;
y2 ~ 1@muy2 RIy@1 y1r x1r;
y3 ~ 1@muy3 RIy@1 y2r x2r;
y4 ~ 1@muy4 RIy@1 y3r x3r;
y5 ~ 1@muy5 RIy@1 y4r x4r;
covariances:
x1 ~ y1;
x2 ~ y2;
x3 ~ y3;
x4 ~ y4;
x5 ~ y5',
seed = 90291,
burn = 5000,
iter = 10000
)
output(mymodel)

```

Mulder and Hamaker (2021) describe four extensions of the RI-CLPM. The first introduces a time-invariant covariate predicting the repeated measures variables. The residual variables that appear as predictors in the models are now computed by subtracting out the random intercept plus the effect of Z_1 as follows.

$$X_{1W}^* = X_1 - [RI_X + \alpha_1(Z_1)] \quad X_{2W}^* = X_2 - [RI_X + \alpha_2(Z_1)]$$

$$Y_{1W}^* = Y_1 - [RI_Y + \beta_1(Z_1)] \quad Y_{2W}^* = Y_2 - [RI_Y + \beta_2(Z_1)]$$

The Blimp script for this five-wave analysis is shown below. The binary covariate Z_1 is identified on the **ORDINAL** line to invoke probit regression in the event of missing data.

```

DATA: RICLPM.dat;
VARIABLES: x1:x5 y1:y5 z2 z1;
ORDINAL: z1;
LATENT: RIx RIy;
MODEL:
x1r = x1 - (mux1 + RIx + z1*a1);
x2r = x2 - (mux2 + RIx + z1*a2);
x3r = x3 - (mux3 + RIx + z1*a3);
x4r = x4 - (mux4 + RIx + z1*a4);
y1r = y1 - (muy1 + RIy + z1*b1);
y2r = y2 - (muy2 + RIy + z1*b2);
y3r = y3 - (muy3 + RIy + z1*b3);
y4r = y4 - (muy4 + RIy + z1*b4);
random.intercepts:
RIx ~~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1 z1@a1;
x2 ~ 1@mux2 RIx@1 x1r y1r z1@a2;
x3 ~ 1@mux3 RIx@1 x2r y2r z1@a3;
x4 ~ 1@mux4 RIx@1 x3r y3r z1@a4;
x5 ~ 1@mux5 RIx@1 x4r y4r z1@a5;
y.models:
y1 ~ 1@muy1 RIy@1 z1@b1;
y2 ~ 1@muy2 RIy@1 y1r x1r z1@b2;
y3 ~ 1@muy3 RIy@1 y2r x2r z1@b3;
y4 ~ 1@muy4 RIy@1 y3r x3r z1@b4;
y5 ~ 1@muy5 RIy@1 y4r x4r z1@b5;
covariances:
x1 ~~ y1;
x2 ~~ y2;
x3 ~~ y3;
x4 ~~ y4;
x5 ~~ y5;
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'RICLPM.rda')

mymodel <- rblimp(
  data = RICLPM,
  ordinal = 'z1',
  latent = 'RIx RIy',
  model = '
x1r = x1 - (mux1 + RIx + z1*a1);
x2r = x2 - (mux2 + RIx + z1*a2);
x3r = x3 - (mux3 + RIx + z1*a3);
x4r = x4 - (mux4 + RIx + z1*a4);
y1r = y1 - (muy1 + RIy + z1*b1);
y2r = y2 - (muy2 + RIy + z1*b2);
y3r = y3 - (muy3 + RIy + z1*b3);
y4r = y4 - (muy4 + RIy + z1*b4);
random.intercept:
RIx ~~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1 z1@a1;
x2 ~ 1@mux2 RIx@1 x1r y1r z1@a2;
x3 ~ 1@mux3 RIx@1 x2r y2r z1@a3;
x4 ~ 1@mux4 RIx@1 x3r y3r z1@a4;
x5 ~ 1@mux5 RIx@1 x4r y4r z1@a5;
y.models:
y1 ~ 1@muy1 RIy@1 z1@b1;
y2 ~ 1@muy2 RIy@1 y1r x1r z1@b2;
y3 ~ 1@muy3 RIy@1 y2r x2r z1@b3;
y4 ~ 1@muy4 RIy@1 y3r x3r z1@b4;
y5 ~ 1@muy5 RIy@1 y4r x4r z1@b5;
covariances:
x1 ~~ y1;
x2 ~~ y2;
x3 ~~ y3;
x4 ~~ y4;
x5 ~~ y5',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```


The second extension introduces a time-invariant covariate predicting the random intercept latent variables. Mulder and Hamaker (2021) note that this model is statistically equivalent to one that places equality constraints on the Z_1 slopes from the previous example. The Blimp script for this analysis is shown below.

```

DATA: RICLPM.dat;
VARIABLES: x1:x5 y1:y5 z2 z1;
ORDINAL: z1;
LATENT: RIx RIy;
MODEL:
x1r = x1 - (mux1 + RIx);
x2r = x2 - (mux2 + RIx);
x3r = x3 - (mux3 + RIx);
x4r = x4 - (mux4 + RIx);
y1r = y1 - (muy1 + RIy);
y2r = y2 - (muy2 + RIy);
y3r = y3 - (muy3 + RIy);
y4r = y4 - (muy4 + RIy);
random intercepts:
RIx ~ z1;
RIy ~ z1;
RIx ~~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1;
x2 ~ 1@mux2 RIx@1 x1r y1r;
x3 ~ 1@mux3 RIx@1 x2r y2r;
x4 ~ 1@mux4 RIx@1 x3r y3r;
x5 ~ 1@mux5 RIx@1 x4r y4r;
y.models:
y1 ~ 1@muy1 RIy@1;
y2 ~ 1@muy2 RIy@1 y1r x1r;
y3 ~ 1@muy3 RIy@1 y2r x2r;
y4 ~ 1@muy4 RIy@1 y3r x3r;
y5 ~ 1@muy5 RIy@1 y4r x4r;
covariances:
x1 ~~ y1;
x2 ~~ y2;
x3 ~~ y3;
x4 ~~ y4;
x5 ~~ y5;

```

```
SEED: 90291;  
BURN: 5000;  
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'RICLPM.rda')  
  
mymodel <- rblimp(  
  data = RICLPM,  
  latent = 'RIx RIy',  
  ordinal = 'z1',  
  model = '  
    x1r = x1 - (mux1 + RIx);  
    x2r = x2 - (mux2 + RIx);  
    x3r = x3 - (mux3 + RIx);  
    x4r = x4 - (mux4 + RIx);  
    y1r = y1 - (muy1 + RIy);  
    y2r = y2 - (muy2 + RIy);  
    y3r = y3 - (muy3 + RIy);  
    y4r = y4 - (muy4 + RIy);  
    random.intercepts:  
    RIx ~ z1;  
    RIy ~ z1;  
    RIx ~~ RIy;  
    x.models:  
    x1 ~ 1@mux1 RIx@1;  
    x2 ~ 1@mux2 RIx@1 x1r y1r;  
    x3 ~ 1@mux3 RIx@1 x2r y2r;  
    x4 ~ 1@mux4 RIx@1 x3r y3r;  
    x5 ~ 1@mux5 RIx@1 x4r y4r;  
    y.models:  
    y1 ~ 1@muy1 RIy@1;  
    y2 ~ 1@muy2 RIy@1 y1r x1r;  
    y3 ~ 1@muy3 RIy@1 y2r x2r;  
    y4 ~ 1@muy4 RIy@1 y3r x3r;  
    y5 ~ 1@muy5 RIy@1 y4r x4r;  
    covariances:  
    x1 ~~ y1;  
    x2 ~~ y2;  
    x3 ~~ y3;  
    x4 ~~ y4;
```

```

x5 ~ y5',
seed = 90291,
burn = 5000,
iter = 10000
)
output(mymodel)

```

The third and fourth extensions feature a distal outcome, Z_2 . This outcome can be regressed on the random intercepts and/or the within-person predictors. The script below features the random intercepts predicting the distal outcome.

```

DATA: RICLPM.dat;
VARIABLES: x1:x5 y1:y5 z2 z1;
LATENT: RIx RIy;
MODEL:
x1r = x1 - (mux1 + RIx);
x2r = x2 - (mux2 + RIx);
x3r = x3 - (mux3 + RIx);
x4r = x4 - (mux4 + RIx);
y1r = y1 - (muy1 + RIy);
y2r = y2 - (muy2 + RIy);
y3r = y3 - (muy3 + RIy);
y4r = y4 - (muy4 + RIy);
random.intercepts:
RIx ~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1;
x2 ~ 1@mux2 RIx@1 x1r y1r;
x3 ~ 1@mux3 RIx@1 x2r y2r;
x4 ~ 1@mux4 RIx@1 x3r y3r;
x5 ~ 1@mux5 RIx@1 x4r y4r;
y.models:
y1 ~ 1@muy1 RIy@1;
y2 ~ 1@muy2 RIy@1 y1r x1r;
y3 ~ 1@muy3 RIy@1 y2r x2r;
y4 ~ 1@muy4 RIy@1 y3r x3r;
y5 ~ 1@muy5 RIy@1 y4r x4r;
distal.outcome:
z2 ~ RIx RIy;
SEED: 90291;

```

```
BURN: 5000;  
ITER: 10000;
```

The corresponding rblimp script is as follows.

```
library(rblimp)  
load(file = 'RICLPM.rda')  
  
mymodel <- rblimp(  
  data = RICLPM,  
  latent = 'RIx RIy',  
  model = '  
x1r = x1 - (mux1 + RIx);  
x2r = x2 - (mux2 + RIx);  
x3r = x3 - (mux3 + RIx);  
x4r = x4 - (mux4 + RIx);  
y1r = y1 - (muy1 + RIy);  
y2r = y2 - (muy2 + RIy);  
y3r = y3 - (muy3 + RIy);  
y4r = y4 - (muy4 + RIy);  
random.intercepts:  
RIx ~~ RIy;  
x.models:  
x1 ~ 1@mux1 RIx@1;  
x2 ~ 1@mux2 RIx@1 x1r y1r;  
x3 ~ 1@mux3 RIx@1 x2r y2r;  
x4 ~ 1@mux4 RIx@1 x3r y3r;  
x5 ~ 1@mux5 RIx@1 x4r y4r;  
y.models:  
y1 ~ 1@muy1 RIy@1;  
y2 ~ 1@muy2 RIy@1 y1r x1r;  
y3 ~ 1@muy3 RIy@1 y2r x2r;  
y4 ~ 1@muy4 RIy@1 y3r x3r;  
y5 ~ 1@muy5 RIy@1 y4r x4r;  
distal.outcome:  
z2 ~ RIx RIy',  
  seed = 90291,  
  burn = 5000,  
  iter = 10000  
)  
output(mymodel)
```

The script below instead shows the within-person variables predicting the distal outcome.

```

DATA: RICLPM.dat;
VARIABLES: x1:x5 y1:y5 z2 z1;
LATENT: RIx RIy;
MODEL:
x1r = x1 - (mux1 + RIx);
x2r = x2 - (mux2 + RIx);
x3r = x3 - (mux3 + RIx);
x4r = x4 - (mux4 + RIx);
x5r = x5 - (mux5 + RIx);
y1r = y1 - (muy1 + RIy);
y2r = y2 - (muy2 + RIy);
y3r = y3 - (muy3 + RIy);
y4r = y4 - (muy4 + RIy);
y5r = y5 - (muy5 + RIy);
random.intercepts:
RIx ~~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1;
x2 ~ 1@mux2 RIx@1 x1r y1r;
x3 ~ 1@mux3 RIx@1 x2r y2r;
x4 ~ 1@mux4 RIx@1 x3r y3r;
x5 ~ 1@mux5 RIx@1 x4r y4r;
y.models:
y1 ~ 1@muy1 RIy@1;
y2 ~ 1@muy2 RIy@1 y1r x1r;
y3 ~ 1@muy3 RIy@1 y2r x2r;
y4 ~ 1@muy4 RIy@1 y3r x3r;
y5 ~ 1@muy5 RIy@1 y4r x4r;
covariances:
x1 ~~ y1;
x2 ~~ y2;
x3 ~~ y3;
x4 ~~ y4;
x5 ~~ y5;
distal.outcome:
z2 ~ x1r y1r x2r y2r x3r y3r x4r y4r x5r y5r;
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'RICLPM.rda')

mymodel <- rblimp(
  data = RICLPM,
  latent = 'RIx RIy',
  model = '
x1r = x1 - (mux1 + RIx);
x2r = x2 - (mux2 + RIx);
x3r = x3 - (mux3 + RIx);
x4r = x4 - (mux4 + RIx);
x5r = x5 - (mux5 + RIx);
y1r = y1 - (muy1 + RIy);
y2r = y2 - (muy2 + RIy);
y3r = y3 - (muy3 + RIy);
y4r = y4 - (muy4 + RIy);
y5r = y5 - (muy5 + RIy);
random.intercepts:
RIx ~~ RIy;
x.models:
x1 ~ 1@mux1 RIx@1;
x2 ~ 1@mux2 RIx@1 x1r y1r;
x3 ~ 1@mux3 RIx@1 x2r y2r;
x4 ~ 1@mux4 RIx@1 x3r y3r;
x5 ~ 1@mux5 RIx@1 x4r y4r;
y.models:
y1 ~ 1@muy1 RIy@1;
y2 ~ 1@muy2 RIy@1 y1r x1r;
y3 ~ 1@muy3 RIy@1 y2r x2r;
y4 ~ 1@muy4 RIy@1 y3r x3r;
y5 ~ 1@muy5 RIy@1 y4r x4r;
covariances:
x1 ~~ y1;
x2 ~~ y2;
x3 ~~ y3;
x4 ~~ y4;
x5 ~~ y5;
distal.outcome:
z2 ~ x1r y1r x2r y2r x3r y3r x4r y4r x5r y5r',
  seed = 90291,
  burn = 5000,

```

```
    iter = 10000  
  )  
output(mymodel)
```

6 Multilevel Model Analysis Examples

This section illustrates multilevel models in Blimp. In general, it is possible to mix and match features from any examples to create complex analysis models that honor features of the data. Following previous chapters, the examples in this section use a generic notation system where variable names usually consist of an alphanumeric prefix and a numeric suffix (e.g., Y , X_1 , X_1N_1 , D_1 , D_2). The letter Y designates a dependent variable, a D prefix denotes a binary dummy variable, an O prefix indicates an ordinal variable, and an N prefix indicates a multicategorical nominal variable. Additionally, the multilevel examples use a “_i” suffix to denote level-1 variables, “_j” for level-2 variables, and “_k” for level-3 variables (e.g., d_j is a level-2 dummy variable, x_i is a continuous predictor measured at level-1). Blimp determines the levels automatically, so the suffixes are meant as a visual aid for understanding the scripts. Finally, the model equations use “cgm” and “cwc” superscripts to indicate grand and group mean centering, respectively. The following list outlines the examples in this section.

- ❖ 6.1: Random Intercept Model
- ❖ 6.2: Two-Level Fully Conditional Specification Multiple Imputation
- ❖ 6.3: Random Coefficient Model
- ❖ 6.4: Multilevel SEM With Random Coefficients
- ❖ 6.5: Alternate Prior Distributions
- ❖ 6.6 Inspecting Residuals
- ❖ 6.7: Heterogeneous) Within-Cluster Variances
- ❖ 6.8: Location-Scale Model With Heterogeneous Within-Cluster Variation
- ❖ 6.9: Random Effects Predicting a Level-2 Outcome
- ❖ 6.10: Latent Contextual Effect
- ❖ 6.11: Cross-Level Interaction Effect
- ❖ 6.12: 1-1-1 Mediation With Random Slopes
- ❖ 6.13: 1-1-1 Moderated Mediation
- ❖ 6.14: Within- and Between-Level Mediation

- ❖ 6.15: Two-Level Growth Model
- ❖ 6.16: Three-Level Growth Model
- ❖ 6.17: Three-Level SEM Growth Model
- ❖ 6.18: Two-Level MIMIC Measurement Model
- ❖ 6.19: Sampling Weights
- ❖ 6.20: Partially Nested Designs (Singleton Clusters)
- ❖ 6.21: Discrete-Time Survival Model

6.1: Random Intercept Model

This example illustrates a two-level regression model with random intercepts. The regression model is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + \beta_1 X_{1ij}^{cgm} + \beta_2 X_{2ij}^{cgm} + \beta_3 D_{1ij}^{cgm} + \beta_4 X_{3j}^{cgm} + \beta_5 D_{2j} + \varepsilon_{ij}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.1.imp](#) [data7.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all incomplete level-1 variables
- ❖ **ORDINAL** command identifies binary predictors
- ❖ **FIXED** command defines complete predictors
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ Unspecified associations for predictor variables

```
DATA: data7.dat;
VARIABLES: level1id level2id v1_i v2_i d1_i v3_i x1_i v4_i
  v5_i x2_i y_i d2_j x3_j v6_j;
CLUSTERID: level2id;
ORDINAL: d1_i d2_j;
```

```

MISSING: 999;
FIXED: x2_i d2_j;
CENTER: grandmean = x1_i x2_i d1_i x3_j;
MODEL: y_i ~ x1_i x2_i d1_i x3_j d2_j;
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data7.rda')

mymodel <- rblimp(
  data = data7,
  clusterid = 'level2id',
  ordinal = 'd1_i d2_j',
  fixed = 'x2_i d2_j',
  center = 'grandmean = x1_i x2_i d1_i x3_j',
  model = 'y_i ~ x1_i x2_i d1_i x3_j d2_j',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

```

6.2: Two-Level Fully Conditional Specification Multiple Imputation

This example illustrates multilevel fully conditional specification multiple imputation as an approach to getting frequentist inference for the analysis from Example 6.1. The analysis model is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + \beta_1 X_{1ij}^{cgm} + \beta_2 X_{4ij}^{cgm} + \beta_3 D_{2ij}^{cgm} + \beta_4 X_{5j}^{cgm} + \beta_5 D_{3j} + \varepsilon_{ij}$$

Fully conditional specification should be reserved for random intercept analyses, as applying the procedure to models with random coefficients or interaction terms is known to induce bias (Enders et al., 2020; Grund, Lüdke, & Robitzsch, 2016).

Model-based multiple imputation is recommended for such analyses (see Example 6.3). Clicking the links below downloads the Blimp scripts and data for this example,

and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.2.imp](#) [Ex6.2.R](#) [data7.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts (latent group means) for all level-1 variables
- ❖ **ORDINAL** command identifies binary predictors
- ❖ **FIXED** command defines complete predictors
- ❖ **FCS** command specifies fully conditional specification multiple imputation with a saturated model at level-1 and level-2 (unstructured within- and between-cluster covariance matrices)
- ❖ **FCS** command includes all analysis variables
- ❖ **NIMPS** command specifies 20 imputed data sets
- ❖ Setting **CHAINS** equal to **NIMPS** saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```
DATA: data7.dat;  
VARIABLES: level1id level2id v1_i v2_i d1_i v3_i x1_i v4_i  
v5_i x2_i y_i d2_j x3_j v6_j;  
CLUSTERID: level2id;  
ORDINAL: d1_i d2_j;  
MISSING: 999;  
FIXED: x2_i d2_j;  
FCS: y_i x1_i x2_i d1_i x3_j d2_j;  
SEED: 90291;  
BURN: 2000;  
ITER: 10000;  
CHAINS: 20;  
NIMPS: 20;  
SAVE: stacked = imps.dat;
```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed.

VARIABLE ORDER IN IMPUTED DATA:

```
stacked = 'imps.dat'
```

```
imp# level1id level2id v1_i v2_i d1_i v3_i x1_i  
v4_i v5_i x2_i y_i d2_j x3_j v6_j
```

The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp_fcs` to create multiple imputations and the `mitml` package (Grund, Robitzsch, & Lüdtke, 2021) for analysis and pooling. Note that the `MISSING` and `FCS` commands are no longer necessary.. The former is omitted because that information is contained in the R data file. The `FCS` command is replaced by a `variables` parameter that lists the variables to be included in the imputation model. Additionally, the `SAVE` command is no longer necessary because imputations are automatically stored in an `rblimp` list object called `mymodel@imputations`.

```
library(rblimp)  
load(file = 'data7.rda')  
  
mymodel <- rblimp_fcs(  
  data = data7,  
  clusterid = 'level2id',  
  ordinal = 'd1_i d2_j',  
  fixed = 'x2_i d2_j',  
  variables = 'y_i x1_i x2_i d1_i x3_j d2_j',  
  seed = 90291,  
  burn = 2000,  
  iter = 10000,  
  chains = 20,  
  nimps = 20  
)  
output(mymodel)
```

```
# mitml list
implist <- as.mitml(mymodel)

# analysis and pooling with mitml
results <- with(implist,
  lmer('y_i ~ x1_i + x2_i + d1_i + x3_j + d2_j + (1|level2id)', REML = T))
testEstimates(results, extra.pars = T)
```

6.3: Random Coefficient Model

This example illustrates a two-level regression model with random intercepts and random slopes. The analysis model is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + \varepsilon_{ij}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.3a.imp](#) [Ex6.3b.imp](#) [Ex6.3.R](#) [data8.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ **ORDINAL** command identifies a binary predictor
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean and latent group mean centering to predictors
- ❖ **MODEL** command features a random coefficient listed after the vertical pipe
- ❖ Unspecified associations for predictor variables

```
DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
  v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
```

```

ORDINAL: d_j;
MISSING: 999;
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL: y_i ~ x1_i x2_i x3_j d_j | x1_i;
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd_j',
  fixed = 'd_j',
  center = 'groupmean = x1_i;
  grandmean = x2_i x3_j d_j',
  model = 'y_i ~ x1_i x2_i x3_j d_j | x1_i',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```

Blimp can save multiple imputations from any model it estimates. Model-based multiple imputations can be saved for a frequentist analysis by adding the `SAVE` and `NIMPS` commands. The additional syntax highlights are as follows.

- ❖ `CENTER` command grand mean centers predictors in the Bayesian output, but saved imputations are on the original metric
- ❖ `NIMPS` command specifies 20 imputed data sets
- ❖ Setting `CHAINS` equal to `NIMPS` saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ `saveLatent` keyword on the `OPTIONS` line saves the latent group means of the level-1 predictors and the analysis model's random intercept and random slope residuals

- ❖ Imputations are stacked in a single file with an index variable added in the first column

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
              v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
ORDINAL: d_j;
MISSING: 999;
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL: y_i ~ x1_i x2_i x3_j d_j | x1_i;
SEED: 90291;
BURN: 5000;
ITER: 10000;
CHAINS: 20;
NIMPS: 20;
OPTIONS: savelatent;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed. The `savelatent` keyword also saves the latent group means of any level-1 predictors, and these can be used to center variables prior to analyzing the imputations. This example uses X_1 's latent group means, which are referred to by the name `x1_i.mean[level2id]`.

VARIABLE ORDER IN IMPUTED DATA:

```

stacked = 'imps.dat'

imp# level1id level2id x1_i x2_i y_i v1_i v2_i d_j v3_j
v4_j v5_j x3_j v6_j v7_j y_i[level2id] y_i$x1_i[level2id]
x1_i.mean[level2id] x2_i.mean[level2id]

```

The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp` to create multiple imputations and the `mitml` package (Grund,

Robitzsch, & Lüdtke, 2021) for analysis and pooling. Note that the `SAVE` command and `savelatent` keyword on the `OPTIONS` line are no longer necessary because imputations and latent variable scores are automatically stored in an `rblimp` list object called `mymodel@imputations`. The pooled estimates are numerically equivalent to the Bayesian results.

```
library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd_j',
  fixed = 'd_j',
  center = 'groupmean = x1_i;
  grandmean = x2_i x3_j d_j',
  model = 'y_i ~ x1_i x2_i x3_j d_j | x1_i',
  seed = 90291,
  burn = 5000,
  iter = 10000,
  chains = 20,
  nimps = 20
)
output(mymodel)

# inspect variable names
names(mymodel@imputations[[1]])

# mitml list
implist <- as.mitml(mymodel)

# pooled grand means
mean_x2 <- mean(unlist(lapply(implist, function(data) mean(data$x2_i))))
mean_x3 <- mean(unlist(lapply(implist, function(data) mean(data$x3_j))))
mean_d <- mean(unlist(lapply(implist, function(data) mean(data$d_j))))

# center at latent cluster means
for (i in 1:length(implist)) {
  implist[[i]]$x1cwc_i <- implist[[i]]$x1_i - implist[[i]]$x1_i.mean.level2id.
}

# analysis and pooling with mitml
results <- with(implist, lmer('y_i ~ x1cwc_i + I(x2_i - mean_x2)
  + I(x3_j - mean_x3) + I(d_j - mean_d) + (1 + x1cwc_i|level2id)', REML = T)
)
```



```
testEstimates(results, extra.pars = T)
```

6.4: Multilevel SEM With Random Coefficients

This example illustrates a two-level regression model with random intercepts and random slopes. The analysis model is the same as Example 6.3, which is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + \varepsilon_{ij}$$

The model is cast as a multilevel structural equation model with a pair of normally distributed level-2 latent variables representing the random intercepts and slopes. The level-1 and level-2 models are as follows.

$$Y_{ij} = \beta_{0j} + \beta_{1j} X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \varepsilon_{ij}$$

$$\beta_{0j} = \beta_0 + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + b_{0j}$$

$$\beta_{1j} = \beta_1 + b_{1j}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.4.imp](#) [data8.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies level-2 and level-3 identifiers (order doesn't matter), automatically inducing random intercepts for all level-1 and level-2 variables
- ❖ **ORDINAL** command identifies binary predictors
- ❖ **FIXED** command defines complete predictors
- ❖ **CENTER** command applies grand mean and latent group mean centering to predictors
- ❖ **LATENT** command defines two between-cluster latent variables representing the random intercepts and slopes

- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command estimates the random intercept and slope means
- ❖ MODEL command sets the intercept of the regression equation equal to the level-2 latent mean (i.e., `1@beta0_j`)
- ❖ MODEL command omits the random coefficient listed after the vertical pipe
- ❖ MODEL command sets the random predictor's slope equal to the random coefficient (i.e., `x1_i@beta1_j`)
- ❖ MODEL command specifies correlation between random intercepts and random slopes (level-2 latent variables)

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
              v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
ORDINAL: d_j;
MISSING: 999;
LATENT: level2id = beta0_j beta1_j;
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL:
  level2.model:
  beta0_j ~ 1 x3_j d_j;
  beta1_j ~ 1;
  beta0_j ~~ beta1_j;
  level1.model:
  y_i ~ 1@beta0_j x1_i@beta1_j x2_i;
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',

```

```

ordinal = 'd_j',
latent = 'level2id = beta0_j beta1_j',
fixed = 'd_j',
center = 'groupmean = x1_i;
grandmean = x2_i x3_j d_j',
model = '
level2.model:
beta0_j ~ 1 x3_j d_j;
beta1_j ~ 1;
beta0_j ~~ beta1_j;
level1.model:
y_i ~ 1@beta0_j x1_i@beta1_j x2_i',
seed = 90291,
burn = 5000,
iter = 10000
)
output(mymodel)

```

The random effect parameter estimates no longer appear on the same table when using the multilevel SEM specification. Rather, each equation has its own summary table. For example, the latent variable summary tables are shown below.

OUTCOME MODEL ESTIMATES:

Summaries based on 10000 iterations using 2 chains.

level2.model block:

Latent Variable: beta0_j

Grand Mean Centered: d1_j x7_j

Parameters	Median	StdDev	2.5%	97.5%

Variances:				
Residual Var.	0.617	0.082	0.482	0.807
Coefficients:				
Intercept	4.167	0.072	4.024	4.309
x7_j	0.050	0.068	-0.082	0.184
d1_j	-0.077	0.140	-0.350	0.201

Standardized Coefficients:

x7_j	0.065	0.087	-0.105	0.233
d1_j	-0.048	0.086	-0.215	0.124
Proportion Variance Explained				
by Coefficients	0.016	0.020	0.001	0.074
by Residual Variation	0.984	0.020	0.926	0.999

Latent Variable: beta1_j				
Parameters	Median	StdDev	2.5%	97.5%

Variances:				
Residual Var.	0.020	0.006	0.011	0.034
Coefficients:				
Intercept	-0.094	0.020	-0.133	-0.055
Proportion Variance Explained				
by Coefficients	0.000	0.000	0.000	0.000
by Residual Variation	1.000	0.000	1.000	1.000

Covariance Matrix: beta0_j beta1_j				
Parameters	Median	StdDev	2.5%	97.5%

Covariances:				
Cov(beta0_j,beta1_j)	0.013	0.016	-0.017	0.047
Correlations:				
Cor(beta0_j,beta1_j)	0.122	0.138	-0.151	0.387

6.5: Alternate Prior Distributions for Random Effect Covariance Matrix

This example illustrates how to examine the influence of different prior distributions on the level-2 covariance matrix of the random effects. The analysis model is the following two-level regression with random intercepts and random slopes.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \varepsilon_{ij}$$

The between-cluster covariance matrix of the random effects is a 2 by 2 matrix in this example. Blimp offers three “off-the-shelf” inverse Wishart priors for the covariance matrix, and it is also possible to use a so-called separation strategy that applies distinct priors to variances and the intercept-slope correlation. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.5a.prior2.imp](#) [Ex6.5b.prior1.imp](#) [Ex6.5c.prior3.imp](#)

[Ex6.5d.separation.imp](#) [data8.dat](#)

Considering the inverse Wishart options, the default `prior2` setting is less informative because it subtracts the number of dimensions plus 1 from the degrees of freedom, and it adds nothing to the sum of squares and cross-products; `prior1` is more informative because it adds the number of dimensions plus 1 to the degrees of freedom, and it adds an identity matrix to the sum of squares and cross-products; `prior3` adds zero degrees of freedom and adds zero to the sums of squares. The code block below shows the default specification, the syntax highlights for which are as follows.

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `CENTER` command applies grand mean and latent group mean centering
- ❖ `MODEL` command features a random coefficient listed after the vertical pipe
- ❖ Unspecified associations for predictor variables
- ❖ `prior2` keyword on the `OPTIONS` line (optional) specifies the default inverse Wishart prior

```
DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
              v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
MISSING: 999;
```

```

CENTER: groupmean = x1_i; grandmean = x2_i;
MODEL: y_i ~ x1_i x2_i | x1_i;
SEED: 90291;
BURN: 10000;
ITER: 10000;
OPTIONS: prior2;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  center = 'groupmean = x1_i;
  grandmean = x2_i',
  model = 'y_i ~ x1_i x2_i | x1_i',
  seed = 90291,
  burn = 10000,
  iter = 10000,
  options = 'prior2'
)
output(mymodel)

```

Similarly, the code block below shows the specification for the more informative `prior1` inverse Wishart option.

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
  v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
MISSING: 999;
CENTER: groupmean = x1_i; grandmean = x2_i;
MODEL: y_i ~ x1_i x2_i | x1_i;
SEED: 90291;
BURN: 10000;
ITER: 10000;
OPTIONS: prior1;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  center = 'groupmean = x1_i;
  grandmean = x2_i',
  model = 'y_i ~ x1_i x2_i | x1_i',
  seed = 90291,
  burn = 10000,
  iter = 10000,
  options = 'prior1'
)
output(mymodel)

```

Comparing the magnitude of the point estimates provides a gauge about the prior distribution's impact. The output table for the default `prior2` specification is shown immediately below, and the second output table shows the results from the more informative `prior1` specification.

```

# prior2

```

Parameters	Median	StdDev	2.5%	97.5%

Variances:				
L2 : Var(Intercept)	0.613	0.083	0.478	0.804
L2 : Cov(x1.i,Intercept)	0.016	0.016	-0.014	0.048
L2 : Var(x1.i)	0.020	0.006	0.010	0.034
Residual Var.	0.358	0.011	0.337	0.381
...				
Proportion Variance Explained				
by Fixed Effects	0.048	0.009	0.032	0.065
by Level-2 Random Intercepts	0.587	0.033	0.524	0.653
by Level-2 Random Slopes	0.021	0.006	0.011	0.036
by Level-1 Residual Variation	0.343	0.027	0.288	0.395

# prior1				
Parameters	Median	StdDev	2.5%	97.5%

Variances:				
L2 : Var(Intercept)	0.591	0.078	0.464	0.771
L2 : Cov(x1.i,Intercept)	0.017	0.018	-0.017	0.053
L2 : Var(x1.i)	0.039	0.007	0.028	0.056
Residual Var.	0.354	0.011	0.333	0.377
...				
Proportion Variance Explained				
by Fixed Effects	0.048	0.009	0.033	0.068
by Level-2 Random Intercepts	0.569	0.033	0.505	0.635
by Level-2 Random Slopes	0.041	0.008	0.028	0.059
by Level-1 Residual Variation	0.341	0.026	0.289	0.392

The default prior 2's random slope variance is roughly half as large as that of the more informative prior (0.020 vs. 0.039), and the two estimates differed by about 2.7 posterior standard deviation units (a very large difference). As a proportion of the total variance, the R^2 effect sizes attributable to the random slopes (Rights & Sterba, 2019) were also quite different (2.1% vs. 4.1%).

The separation strategy (Barnard, McCulloch, & Meng, 2000; Liu, Zhang, & Grimm, 2016) assigns distinct priors to the diagonal and off-diagonal elements of the covariance matrix. An analogous strategy can be implemented in Blimp by specifying the random intercepts and slopes as a pair of level-2 latent variables. The focal model is cast as a multilevel structural equation model with a pair of normally distributed level-2 latent variables representing the random intercepts and slopes. The focal model features these latent variables as predictors, as shown below.

$$\beta_{0j} = \beta_0 + b_{0j}$$

$$\beta_{1j} = \beta_1 + b_{1j}$$

$$Y_{ij} = \beta_{0j} + \beta_{1j}X_{1ij}^{cwc} + \beta_2X_{2ij}^{cgm} + \varepsilon_{ij}$$

Note that the coefficient of the random slope predictor is implicitly fixed to one in this specification.

This specification assigns separate inverse gamma priors to the random intercept and slope variances, and it specifies a beta prior distribution to their correlation. Blimp uses a multilevel extension of the procedure described in Merkle and Rosseel (2018). Computer simulation studies suggest that the separation strategy gives more accurate estimates of the variance components, although the correlation estimate may be attenuated when the number of level-2 units is small (Keller & Enders, 2021). The unique syntax highlights for the code block are as follows.

- ❖ **LATENT** command defines two between-cluster latent variables representing the random intercepts and slopes
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command estimates the random intercept and slope means
- ❖ **MODEL** command sets the intercept of the regression equation equal to the level-2 latent mean (**1@beta0.j**)
- ❖ **MODEL** command omits the random coefficient listed after the vertical pipe
- ❖ **MODEL** command sets the random predictor's slope equal to the random coefficient (**x1_i@beta1_j**)
- ❖ **MODEL** command specifies correlation between random intercepts and random slopes (level-2 latent variables)
- ❖ **OPTIONS** command lists the `use_phantom` keyword to invoke a phantom variable specification that assigns distinct priors to latent variable variances and their correlation

```
DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
              v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
MISSING: 999;
```

```

LATENT: level2id = beta0_j beta1_j;
CENTER: groupmean = x1_i; grandmean = x2_i;
MODEL:
  latent.variables:
  beta0_j ~ 1;
  beta1_j ~ 1;
  beta0_j ~~ beta1_j;
  focal.model:
  y_i ~ 1@beta0_j x1_i@beta1_j x2_i;
SEED: 90291;
BURN: 10000;
ITER: 10000;
OPTIONS: use_phantom;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  latent = 'level2id = beta0_j beta1_j',
  center = 'groupmean = x1_i;
  grandmean = x2_i',
  model = '
  beta1_j ~ 1;
  beta0_j ~~ beta1_j;
  focal.model:
  y_i ~ 1@beta0_j x1_i@beta1_j x2_i',
  seed = 90291,
  burn = 10000,
  iter = 10000,
  options = 'use_phantom'
)
output(mymodel)

```

The random effect parameter estimates no longer appear on the same table when employing the separation strategy because the random intercepts and slopes are latent variables with their own equations and summary tables. The analysis model

table shows the random intercept variance, and the level-2 latent variable's (random slope) variance and correlation appear in separate tables.

```
# separation strategy
```

Parameters	Median	StdDev	2.5%	97.5%
Latent Variable: beta0_j				

Variances:				
Residual Var.	0.605	0.081	0.472	0.792
Coefficients:				
Intercept	4.170	0.071	4.031	4.309
Proportion Variance Explained				
by Coefficients	0.000	0.000	0.000	0.000
by Residual Variation	1.000	0.000	1.000	1.000

Latent Variable: beta1_j				

Variances:				
Residual Var.	0.019	0.005	0.011	0.031
Coefficients:				
Intercept	-0.094	0.020	-0.132	-0.055
Proportion Variance Explained				
by Coefficients	0.000	0.000	0.000	0.000
by Residual Variation	1.000	0.000	1.000	1.000

```
Phantom Variable Correlations:
```

Parameters	Median	StdDev	2.5%	97.5%

beta0_j ~ beta1_j	0.112	0.123	-0.127	0.348

...				

```

Outcome Variable: y_i
Grand Mean Centered: x2_i
Group Mean Centered: x1.i

```

Parameters	Median	StdDev	2.5%	97.5%

Variances:				
Residual Var.	0.358	0.011	0.337	0.381
Coefficients:				
beta0_j	@1.000	---	---	---
x2_i	0.087	0.008	0.071	0.102
x1.i*beta1_j	@1.000	---	---	---
Standardized Coefficients:				
x2_i	0.291	0.025	0.242	0.339
x1.i*beta1_j	0.284	0.025	0.235	0.334
Proportion Variance Explained				
by Coefficients	0.177	0.020	0.140	0.218
by Residual Variation	0.823	0.020	0.782	0.860

6.6: Inspecting Residuals

This example illustrates how to inspect the level-1 and level-2 residuals (random effects) from a two-level regression model with random intercepts and random slopes. The analysis model, shown below, is the same as the one from Example 6.4.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \varepsilon_{ij}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.6.imp](#) [Ex6.6.R](#) [data8.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ **CENTER** command applies grand mean and latent group mean centering to predictors in the Bayesian output, but saved imputations are on the original metric
- ❖ **MODEL** command features a random coefficient listed after the vertical pipe
- ❖ Unspecified associations for predictor variables
- ❖ **savelatent** keyword on the **OPTIONS** line saves the latent group means of the level-1 predictors and the analysis model's random intercept and random slope residuals
- ❖ **saveresidual** keyword on the **OPTIONS** line saves level-1 residuals
- ❖ **NIMPS** command specifies 20 imputed data sets
- ❖ Setting **CHAINS** equal to **NIMPS** saves one data set from the final iteration of each MCMC chain (avoids autocorrelated imputations)
- ❖ Imputations are stacked in a single file with an index variable added in the first column

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
  v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
MISSING: 999;
CENTER: groupmean = x1_i; grandmean = x2_i;
MODEL: y_i ~ x1_i x2_i | x1_i;
SEED: 90291;
BURN: 5000;
ITER: 10000;
CHAINS: 20;
NIMPS: 20;
OPTIONS: savelatent saveresidual;
SAVE: stacked = imps.dat;

```

Blimp lists the order of the variables in the imputed data sets at the bottom of the output file, and all variables in the input file appear in the output file regardless of whether they were imputed. The latent group means, random effects, and level-1 residuals are appended to the end of the file. Latent group means are designated by appending the level-2 identifier in square brackets to the end of a predictor variable's

name (e.g., `x1_i.mean[level2id]` and `x2_i.mean[level2id]`). The analysis model's random intercepts are denoted by appending the level-2 identifier in square brackets to the end of an outcome variable's name (e.g., `y_i[level2id]`). Random slope residuals are indicated by joining the outcome and random predictor variables with a \$ sign (e.g., `y_i$x1_i[level2id]`). Finally, level-1 residuals are indicated by appending `.residual` to the end of the outcome variable's name (e.g., `y_i.residual`).

VARIABLE ORDER IN IMPUTED DATA:

```
stacked = 'imps.dat'

imp# level1id level2id x1_i x2_i y_i v1_i v2_i d_j v3_j
v4_j v5_j x3_j v6_j v7_j y_i[level2id] y_i$x1_i[level2id]
x1_i.mean[level2id] x2_i.mean[level2id] y_i.residual
```

The imputed data sets can be analyzed in other software packages.

R provides an easy platform for analyzing multiple imputations. To illustrate, R script below uses `rblimp` to create multiple imputations for graphing. Note that the `SAVE` command and `saveLatent` keyword on the `OPTIONS` line are no longer necessary because imputations and latent variable scores are automatically stored in an `rblimp` list object called `mymodel@imputations`.

```
library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd_j',
  fixed = 'd_j',
  center = 'groupmean = x1_i;
  grandmean = x2_i x3_j d_j',
  model = 'y_i ~ x1_i x2_i x3_j d_j | x1_i',
  seed = 90291,
  burn = 5000,
  iter = 10000,
  chains = 20,
  nimps = 20
```

```

)
output(mymodel)

# inspect variable names in imputed data
names(mymodel@imputations[[1]])

# stack list of imputed data sets
dat2plot <- do.call(rbind, mymodel@imputations)
# plot random intercepts
hist(dat2plot$y_i.level2id., breaks = 50)
plot(density(dat2plot$y_i.level2id.))
# plot random slopes
hist(dat2plot$y_i.x1_i.level2id., breaks = 50)
plot(density(dat2plot$y_i.x1_i.level2id.))

```

6.7: Heterogeneous Within-Cluster Variation

This example illustrates a two-level regression model with random intercepts and slopes and heterogeneous within-cluster variances. The analysis model below is the same one as Example 6.3, but the variance of the within-cluster residuals differs across clusters.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + \varepsilon_{ij}$$

Blimp provides two methods for modeling heterogeneous within-cluster variation. The first is an approach described by Kasim and Raudenbush (1998). Their model views cluster-specific variances as a level-2 variable. Unlike the location-scale model in Example 6.8, the Kasim and Raudenbush method does not allow random variation to correlate with or link to other level-2 variables and random effects. Thus, the intent of this model is to simply adjust for heteroscedasticity in the same spirit as robust standard errors.

Clicking the links below downloads the Blimp scripts and data for these examples, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.7.imp](#) [data8.dat](#)

The code block below shows the setup for the Kasim and Raudenbush (1998) approach to modeling heterogeneous variation. The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `ORDINAL` command identifies a binary predictor
- ❖ `FIXED` command identifies a complete predictor
- ❖ `CENTER` command applies grand mean and latent group mean centering to predictors
- ❖ `MODEL` command features a random coefficient listed after the vertical pipe
- ❖ Unspecified associations for predictor variables
- ❖ `hev` keyword on `OPTIONS` line specifies heterogeneous within-cluster variances (Kasim & Raudenbush, 1998)

```
DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
              v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
ORDINAL: d_j;
MISSING: 999;
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL: y_i ~ x1_i x2_i x3_j d_j | x1_i;
SEED: 90291;
BURN: 5000;
ITER: 10000;
OPTIONS: hev;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
```



```

clusterid = 'level2id',
ordinal = 'd_j',
fixed = 'd_j',
center = 'groupmean = x1_i;
grandmean = x2_i x3_j d_j',
model = 'y_i ~ x1_i x2_i x3_j d_j | x1_i',
seed = 90291,
burn = 5000,
iter = 10000,
options = 'hev'
)
output(mymodel)

```

MCMC estimation yields an estimate of the variation within each cluster. To convey the magnitude of the variational differences, Blimp computes the mean and quartiles of the variance distribution and includes these summaries on the output. The output excerpt below shows part of the main summary table from the example.

OUTCOME MODEL ESTIMATES:

Summaries based on 10000 iterations using 2 chains.

Outcome Variable: y_i

Grand Mean Centered: d1_j x2_i x7_j

Group Mean Centered: x1_i

Parameters	Median	StdDev	2.5%	97.5%

Variances:				
L2 : Var(Intercept)	0.648	0.088	0.507	0.851
L2 : Cov(x1_i,Intercept)	0.026	0.014	-0.000	0.056
L2 : Var(x1_i)	0.014	0.006	0.006	0.028
Heterogeneity Index	0.207	0.036	0.149	0.288
Q25% Residual Var.	0.188	0.011	0.168	0.211
Q50% Residual Var.	0.296	0.016	0.267	0.328
Mean Residual Var.	0.373	0.016	0.345	0.406
Q75% Residual Var.	0.476	0.028	0.426	0.534
...				

6.8: Location–Scale Model With Heterogeneous Within-Cluster Variation

This example illustrates a two-level location-scale model with random intercepts, random slopes, and random heterogeneous within-cluster variances. Hedeker, Mermelstein, and Demirtas (2008) and more recently McNeish (2021) describe the model in detail. The analysis model below is the same one as Example 6.4, but the variance of the within-cluster residuals differs across clusters.

The analysis model is the same as Example 6.3, which is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + \varepsilon_{ij}$$

Following Example 6.4, the model is cast as a multilevel structural equation model with a pair of normally distributed level-2 latent variables representing the random intercepts and slopes. The level-1 and level-2 models are as follows.

$$Y_{ij} = \beta_{0j} + \beta_{1j} X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \varepsilon_{ij}$$

$$\beta_{0j} = \beta_0 + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + b_{0j}$$

$$\beta_{1j} = \beta_1 + b_{1j}$$

A location-scale model expresses the natural log of the within-cluster variance as a level-2 latent variable (random effect). The mean and variance of this latent variable encode the typical amount of variation and between-cluster differences in the within-cluster variation (on the logarithmic metric). The scale model has both a within-cluster and between-cluster component, and predictors can be added at each level. The equations below add a predictor at each level.

$$\gamma_{0j} = \gamma_0 + \gamma_2 D_j^{cgm} + g_{0j}$$

$$\ln(\sigma_{\varepsilon_{ij}}^2) = \gamma_{0j} + \gamma_1 X_{1ij}^{cwc}$$

This approach allows within-cluster variation to function as both an outcome and a predictor of distal level-2 outcomes. It is typical to allow the logarithmic latent variable to correlate with the random intercepts and slopes from the focal model.

Clicking the links below downloads the Blimp scripts and data for these examples, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.8.imp](#) [data8.dat](#)

The code block below shows the basic setup for a location-scale model where observation-level variation is a function of a level-1 and level-2 predictor and a level-2 random effect. The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `ORDINAL` command identifies a binary predictor
- ❖ `FIXED` command defines a complete predictor
- ❖ `LATENT` command initializes three level-2 latent variables that represent the random intercepts, random slopes, and random within-cluster variances on the logarithmic scale
- ❖ `CENTER` command applies grand mean and latent group mean centering to predictors
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ `MODEL` command estimates the latent variable means
- ❖ `MODEL` command sets the intercept of the regression equation equal to the level-2 latent mean (`1@beta0_j`)
- ❖ `MODEL` command omits the random coefficient listed after the vertical pipe
- ❖ `MODEL` command sets the random predictor's slope equal to the random coefficient (`x1_i@beta1_j`)
- ❖ `MODEL` command includes a variance model for the outcome using the `var(y1_i)` function

- ❖ MODEL command sets the intercept of the log-variance model equal to the level-2 latent mean (e.g., $\text{var}(y_{1_i}) \sim 1@logvar_j$)
- ❖ MODEL command specifies correlations among all random effects

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
              v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
ORDINAL: d_j;
MISSING: 999;
LATENT: level2id = beta0_j beta1_j logvar_j;
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL:
  level2.model:
  beta0_j ~ 1 x3_j d_j;
  beta1_j ~ 1;
  beta0_j ~~ beta1_j;
  level1.model:
  y_i ~ 1@beta0_j x1_i@beta1_j x2_i;
  variance.model:
  logvar_j ~ 1 d_j;
  var(y_i) ~ 1@logvar_j x1_i;
  logvar_j ~~ beta0_j beta1_j;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd_j',
  latent = 'level2id = beta0_j beta1_j logvar_j',
  fixed = 'd_j',
  center = 'groupmean = x1_i;
  grandmean = x2_i x3_j d_j',

```

```

model = '
level2.model:
beta0_j ~ 1 x3_j d_j;
beta1_j ~ 1;
beta0_j ~ beta1_j;
level1.model:
y_i ~ 1@beta0_j x1_i@beta1_j x2_i;
variance.model:
logvar_j ~ 1 d_j;
var(y_i) ~ 1@logvar_j x1_i;
logvar_j ~ beta0_j beta1_j',
seed = 90291,
burn = 10000,
iter = 10000
)
output(mymodel)

```

MCMC estimation yields a model-predicted variance for each observation (on the natural log scale). To convey the magnitude of the variational differences, Blimp converts the mean and quartiles of the log-variance distribution to the variance metric and includes these summaries on the output. The output excerpt below shows part of the main summary table from the example.

OUTCOME MODEL ESTIMATES:

Summaries based on 10000 iterations using 2 chains.

...

Outcome Variable: y1_i

Grand Mean Centered: x2_i

Group Mean Centered: x1_i

Parameters	Median	StdDev	2.5%	97.5%

Variances:				
Q25% Residual Var.	0.190	0.011	0.171	0.212
Q50% Residual Var.	0.307	0.015	0.280	0.339
Mean Residual Var.	0.371	0.014	0.344	0.401
Q75% Residual Var.	0.486	0.025	0.440	0.538

Coefficients:				
beta0_j	@1.000	---	---	---
x2_i	0.082	0.008	0.066	0.096
x1_i*beta1_j	@1.000	---	---	---
Standardized Coefficients:				
x2_i	0.273	0.024	0.224	0.321
x1_i*beta1_j	0.257	0.031	0.196	0.318
Proportion Variance Explained				
by Coefficients	0.150	0.021	0.113	0.193
by Residual Variation	0.850	0.021	0.807	0.887

...				

6.9: Random Effects Predicting a Level-2 Outcome

This example illustrates a two-level regression model with random intercepts and random slopes. The focal analysis model is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + \varepsilon_{ij}$$

The random intercepts and random slopes in turn predict a distal outcome, as follows.

$$Y_{2j} = \gamma_0 + \gamma_1 b_{0j} + \gamma_2 b_{1j} + r_i$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.9a.imp](#) [Ex6.9b.imp](#) [data8.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ **ORDINAL** command identifies a binary predictor

- ❖ **RANDOMEFFECT** command defines random intercepts and slopes as level-2 latent variables
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean and latent group mean centering to predictors
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command features a random coefficient listed after the vertical pipe
- ❖ Unspecified associations for predictor variables

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
  v3_j v4_j v5_j x3_j v6_j y2_j;
CLUSTERID: level2id;
ORDINAL: d_j;
MISSING: 999;
RANDOMEFFECT:
beta0_j = y_i | 1 [level2id];
beta1_j = y_i | x1_i [level2id];
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL:
focal.model:
y_i ~ x1_i x2_i x3_j d_j | x1_i;
distal.outcome:
y2_j ~ beta0_j beta1_j x3_j;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd_j',

```

```

randomeffect = 'beta0_j = y_i | 1 [level2id];
beta1_j = y_i | x1_i [level2id]',
fixed = 'd_j',
center = 'groupmean = x1_i;
grandmean = x2_i x3_j d_j',
model = '
focal.model:
y_i ~ x1_i x2_i x3_j d_j | x1_i;
distal.outcome:
y2_j ~ beta0_j beta1_j x3_j',
seed = 90291,
burn = 10000,
iter = 10000
)
output(mymodel)

```

An alternate approach defines a pair of level-2 latent variables that represent the random intercepts and slopes. The setup of this model is identical to the multilevel SEM in Example 6.4.

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
v3_j v4_j v5_j x3_j v6_j y2_j;
CLUSTERID: level2id;
ORDINAL: d_j;
MISSING: 999;
LATENT: level2id = beta0_j beta1_j;
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL:
  level2.model:
beta0_j ~ 1 x3_j d_j;
beta1_j ~ 1;
beta0_j ~~ beta1_j;
  level1.model:
y_i ~ 1@beta0_j x1_i@beta1_j x2_i;
  distal.outcome:
y2_j ~ beta0_j beta1_j x3_j;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```


The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd_j',
  latent = 'level2id = beta0_j beta1_j',
  fixed = 'd_j',
  center = 'groupmean = x1_i;
  grandmean = x2_i x3_j d_j',
  model = '
  level2.model:
  beta0_j ~ 1 x3_j d_j;
  beta1_j ~ 1;
  beta0_j ~~ beta1_j;
  level1.model:
  y_i ~ 1@beta0_j x1_i@beta1_j x2_i;
  distal.outcome:
  y2_j ~ beta0_j beta1_j x3_j',
  seed = 90291,
  burn = 10000,
  iter = 10000
)
output(mymodel)
```

6.10: Latent Contextual Effect Model

This example illustrates a two-level regression model that includes within- and between-cluster slopes for a level-1 predictor and a latent contextual effect (Lüdtke et al., 2008).

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j})(X_{ij}^{cwc}) + \beta_2(\mu_{X_j}^{cgm}) + \varepsilon_{ij}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.10.imp](#) [data8.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `CENTER` command applies grand mean and latent group mean centering to predictors
- ❖ `MODEL` command features a random coefficient listed after the vertical pipe
- ❖ `MODEL` command specifies latent group means as a level-2 predictor with the `.mean` suffix on a level-1 predictor
- ❖ `MODEL` command labels within- and between-cluster slopes
- ❖ `PARAMETERS` command uses labeled quantities to compute latent contextual effect (between- vs. within-cluster slope difference)

```

DATA: data8.dat;
VARIABLES: level1id level2id v1_i x_i y_i v2_i:v10_i;
CLUSTERID: level2id;
MISSING: 999;
CENTER: groupmean = x_i; grandmean = x_i.mean;
MODEL:
y_i ~ x_i@beta_w x_i.mean@beta_b | x_i;
PARAMETERS:
contextual = beta_b - beta_w;
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  center = 'groupmean = x_i;
  grandmean = x_i.mean',

```

```

model = 'y_i ~ x_i@beta_w x_i.mean@beta_b | x_i',
parameters = 'contextual = beta_b - beta_w',
seed = 90291,
burn = 5000,
iter = 10000
)
output(mymodel)

```

6.11: Cross-Level Interaction Effect

This example illustrates a two-level regression model that includes a cross-level interaction involving a continuous level-1 predictor and a continuous level-2 moderator. The regression model is as follows.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j})(X_{ij}^{cwc}) + \beta_2(M_j^{cgm}) + \beta_3(X_{ij}^{cwc})(M_j^{cgm}) + \varepsilon_{ij}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.11a.imp](#) [Ex6.11b.imp](#) [data1.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `CENTER` command applies grand mean and latent group mean centering to predictors
- ❖ `MODEL` command features a random coefficient listed after the vertical pipe
- ❖ `MODEL` command features a product term
- ❖ `SIMPLE` command produces conditional effects (simple slopes) at different standard deviation units of the continuous moderator
- ❖ Unspecified associations for predictor variables

```

DATA: data1.dat;
VARIABLES: level1id level2id v1_i v2_i y_i x_i v3_i m_j v4_j;
CLUSTERID: level2id;
MISSING: 999;
CENTER: groupmean = x_i; grandmean = m_j;
MODEL:
y_i ~ x_i m_j x_i*m_j | x_i;
SIMPLE: x_i | m_j;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  clusterid = 'level2id',
  center = 'groupmean = x_i',
  grandmean = 'm_j',
  model = 'y_i ~ x_i m_j x_i*m_j | x_i',
  simple = 'x_i | m_j',
  seed = 90291,
  burn = 10000,
  iter = 10000
)
output(mymodel)

```

Alternatively, the analysis can be cast as a multilevel structural equation model. The model setup is similar to Example 6.4. This model does not include an explicit product term. Rather, the cross-level product is represented as a level-2 predictor in the random slope latent variable's equation. The level-1 and level-2 models are as follows.

$$Y_{ij} = \beta_{0j} + \beta_{1j}X_{ij}^{cwc} + \varepsilon_{ij}$$

$$\beta_{0j} = \beta_0 + \beta_2M_j^{cgm} + b_{0j}$$

$$\beta_{1j} = \beta_1 + \beta_3 M_j^{cgm} + b_{1j}$$

The script for the multilevel SEM is shown below.

```

DATA: data1.dat;
VARIABLES: level1id level2id v1_i v2_i y_i x_i v3_i m_j v4_j;
CLUSTERID: level2id;
MISSING: 999;
LATENT: level2id = beta0_j beta1_j;
CENTER: groupmean = x_i; grandmean = m_j;
MODEL:
  level2.model:
  beta0_j ~ 1 m_j;
  beta1_j ~ 1 m_j;
  beta0_j ~~ beta1_j;
  level1.model:
  y_i ~ 1@beta0_j x_i@beta1_j;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  clusterid = 'level2id',
  latent = 'level2id = beta0_j beta1_j',
  center = 'groupmean = x_i;
  grandmean = m_j',
  model = '
  level2.model:
  beta0_j ~ 1 m_j;
  beta1_j ~ 1 m_j;
  beta0_j ~~ beta1_j;
  level1.model:
  y_i ~ 1@beta0_j x_i@beta1_j',
  seed = 90291,
  burn = 10000,
  iter = 10000
)

```

```
output(mymodel)
```

6.12: 1-1-1 Mediation With Random Slopes

This example illustrates a two-level path model that features an indirect effect of two level-1 predictors, both of which are within-cluster centered at their latent group means. The regression models are as follows.

$$X_{ij} = \mu_{X_j} + \varepsilon_{X_{ij}}$$

$$M_{ij} = \mu_{M_j} + \alpha_j(X_{ij} - \mu_{X_j}) + \varepsilon_{M_{ij}}$$

$$Y_{ij} = \mu_{Y_j} + \beta_j(M_{ij} - \mu_{M_j}) + \tau_j'(X_{ij} - \mu_{X_j}) + \varepsilon_{Y_{ij}}$$

$$\mu_{X_j} = \mu_X + u_{X_j}$$

$$\mu_{M_j} = \mu_M + u_{M_j}$$

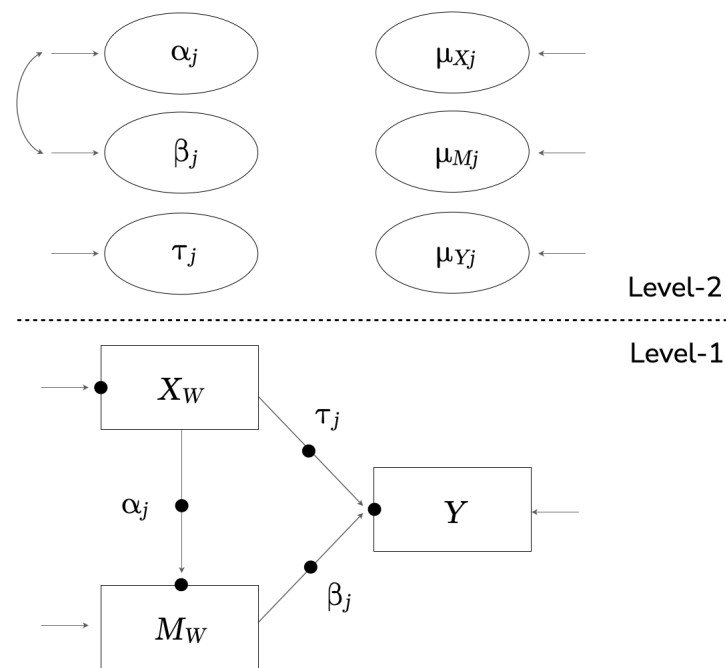
$$\mu_{Y_j} = \mu_Y + u_{Y_j}$$

$$\alpha_j = \mu_\alpha + u_{\alpha_j}$$

$$\beta_j = \mu_\beta + u_{\beta_j}$$

$$\tau_j' = \mu_{\tau'} + u_{\tau_j'}$$

The random slopes are pure within-cluster effects because the predictors are centered at their latent cluster means. The path diagram below shows the model.



The ellipses in the between-cluster model represent the latent group means (i.e., random intercepts) and random slopes. Note that the α and β random slopes are correlated and all other random effects are independent. This model specification follows Yuan and MacKinnon (2009).

The latent variable definitions are the same as those from the multilevel SEM analyses in Example 6.4. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.12.imp](#) [data1.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables

- ❖ LATENT command defines between-cluster latent variables representing the random intercepts and slopes
- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command estimates the random intercept and slope means, using the `->` operator to specify means for all variables that do not require labels
- ❖ MODEL command sets the intercept of the regression equation equal to the level-2 latent mean (e.g., `x_i ~ 1@xmean_j`)
- ❖ MODEL command centers predictors at their latent group means to obtain pure within-cluster variables (e.g., `x_i - xmean_j`)
- ❖ MODEL command omits the random coefficient listed after the vertical pipe
- ❖ MODEL command sets the random predictor's slope equal to the random coefficient (e.g., `(x_i - xmean_j)@alpha_j`)
- ❖ MODEL command specifies the correlation between random slopes
- ❖ PARAMETERS command uses labeled quantities to compute the product of coefficients estimator
- ❖ PARAMETERS command uses the `.totalvar` function to access the variance of the random effects

```

DATA: data1.dat;
VARIABLES: level1id level2id v1_i y_i m_i x_i v2_i v3_j v4_j;
CLUSTERID: level2id;
MISSING: 999;
LATENT: level2id = xmean_j mmean_j ymean_j
           alpha_j beta_j tau_j;
MODEL:
  level2.models:
  1 -> xmean_j mmean_j ymean_j tau_j;
  alpha_j ~ 1@alpha_mean;
  beta_j ~ 1@beta_mean;
  alpha_j ~~ beta_j@ab_cor;
  level1.models:
  x_i ~ 1@xmean_j;
  m_i ~ 1@mmean_j (x_i - xmean_j)@alpha_j;
  y_i ~ 1@ymean_j (m_i - mmean_j)@beta_j (x_i - xmean_j)@tau_j;
PARAMETERS:
  ab_cov = ab_cor * sqrt(alpha_j.totalvar * beta_j.totalvar);
  ab = alpha_mean * beta_mean + ab_cov;

```



```
SEED: 90291;
BURN: 10000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  clusterid = 'level2id',
  latent = 'level2id = xmean_j mmean_j ymean_j alpha_j beta_j tau_j',
  model = '
level2.models:
1 -> xmean_j mmean_j ymean_j tau_j;
alpha_j ~ 1@alpha_mean;
beta_j ~ 1@beta_mean;
alpha_j ~~ beta_j@ab_cor;
level1.models:
x_i ~ 1@xmean_j;
m_i ~ 1@mmean_j (x_i - xmean_j)@alpha_j;
y_i ~ 1@ymean_j (m_i - mmean_j)@beta_j (x_i - xmean_j)@tau_j',
  parameters = 'ab_cov = ab_cor * sqrt(alpha_j.totalvar * beta_j.totalvar);
ab = alpha_mean * beta_mean + ab_cov',
  seed = 90291,
  burn = 10000,
  iter = 10000
)
output(mymodel)
```

6.13: 1-1-1 Moderated Mediation

This example illustrates a two-level path model that features an indirect effect of two level-1 predictors, both of which are within-cluster centered at their latent group means. A level-2 predictor moderates the α path. Any path can be moderated following the procedures from this example. The model equations are identical to Example 6.12 with two exceptions: the level-2 moderator W appears in M 's random intercept equation and α 's random slope equation, as follows.

$$\mu_{M_j} = \mu_M + \gamma_1(W_j) + u_{M_j}$$

$$\alpha_j = \mu_\alpha + \gamma_2(W_j) + u_{\alpha_j}$$

The γ_2 coefficient captures the cross-level moderation of W on the α slope.

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.13.imp](#) [data1.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `LATENT` command defines between-cluster latent variables representing the random intercepts and slopes
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ `MODEL` command estimates the random intercept and slope means, using the `->` operator to specify means for all variables that do not require labels
- ❖ `MODEL` command sets the intercept of the regression equation equal to the level-2 latent mean (e.g., `x_i ~ 1@xmean_j`)
- ❖ `MODEL` command centers predictors at their latent group means to obtain pure within-cluster variables (e.g., `x_i - xmean_j`)
- ❖ `MODEL` command omits the random coefficient listed after the vertical pipe
- ❖ `MODEL` command sets the random predictor's slope equal to the random coefficient (e.g., `(x_i - xmean_j)@alpha_j`)
- ❖ `MODEL` command specifies the correlation between random slopes
- ❖ `PARAMETERS` command uses labeled quantities to compute the product of coefficients estimator
- ❖ `PARAMETERS` command uses the `.totalvar` function to access the variance of the random effects

- ❖ **PARAMETERS** command computes the conditional mediated effect at three values of the moderator

```

DATA: data1.dat;
VARIABLES: level1id level2id v1_i y_i m_i x_i v2_i w_j v3_j;
CLUSTERID: level2id;
MISSING: 999;
LATENT: level2id = xmean_j mmean_j ymean_j
           alpha_j beta_j tau_j;
CENTER: grandmean = w_j;
MODEL:
level2.models:
1 -> xmean_j mmean_j ymean_j tau_j w_j;
mmean_j ~ w_j;
alpha_j ~ 1@alpha_mean w_j@product;
beta_j ~ 1@beta_mean;
alpha_j ~~ beta_j@ab_cor;
level1.models:
x_i ~ 1@xmean_j;
m_i ~ 1@mmean_j (x_i - xmean_j)@alpha_j;
y_i ~ 1@ymean_j (m_i - mmean_j)@beta_j (x_i - xmean_j)@tau_j;
PARAMETERS:
w_stddev = sqrt(w_j.totalvar);
ab_cov = ab_cor * sqrt(alpha_j.totalvar * beta_j.totalvar);
ab_low = (alpha_mean - (product * w_stddev)) *
         beta_mean + ab_cov;
ab_med = alpha_mean * beta_mean + ab_cov;
ab_high = (alpha_mean + (product * w_stddev)) *
          beta_mean + ab_cov;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  clusterid = 'level2id',
  latent = 'level2id = xmean_j mmean_j ymean_j alpha_j beta_j tau_j',

```

```

center = 'grandmean = w_j',
model = '
level2.models:
1 -> xmean_j mmean_j ymean_j tau_j w_j;
mmean_j ~ w_j;
alpha_j ~ 1@alpha_mean w_j@product;
beta_j ~ 1@beta_mean;
alpha_j ~~ beta_j@ab_cor;
level1.models:
x_i ~ 1@xmean_j;
m_i ~ 1@mmean_j (x_i - xmean_j)@alpha_j;
y_i ~ 1@ymean_j (m_i - mmean_j)@beta_j (x_i - xmean_j)@tau_j',
parameters = 'w_stddev = sqrt(w_j.totalvar);
ab_cov = ab_cor * sqrt(alpha_j.totalvar * beta_j.totalvar);
ab_low = (alpha_mean - (product * w_stddev)) *
beta_mean + ab_cov;
ab_med = alpha_mean * beta_mean + ab_cov;
ab_high = (alpha_mean + (product * w_stddev)) *
beta_mean + ab_cov',
seed = 90291,
burn = 10000,
iter = 10000
)
output(mymodel)

```

6.14: Within- and Between-Level Mediation

This example illustrates a two-level path model that features a within-cluster indirect effect involving centered level-1 variables and a between-cluster indirect effect involving latent group means. The analysis expands on Example 6.12 by specifying directed pathways at level-2. The primary regression models are as follows.

$$X_{ij} = \mu_{X_j} + \varepsilon_{X_{ij}}$$

$$M_{ij} = \mu_{M_j} + \alpha_j(X_{ij} - \mu_{X_j}) + \varepsilon_{M_{ij}}$$

$$Y_{ij} = \mu_{Y_j} + \beta_j(M_{ij} - \mu_{M_j}) + \tau_j'(X_{ij} - \mu_{X_j}) + \varepsilon_{Y_{ij}}$$

$$\mu_{X_j} = \mu_X + u_{X_j}$$

$$\mu_{M_j} = I_M + \alpha_B(\mu_{X_j}) + u_{M_j}$$

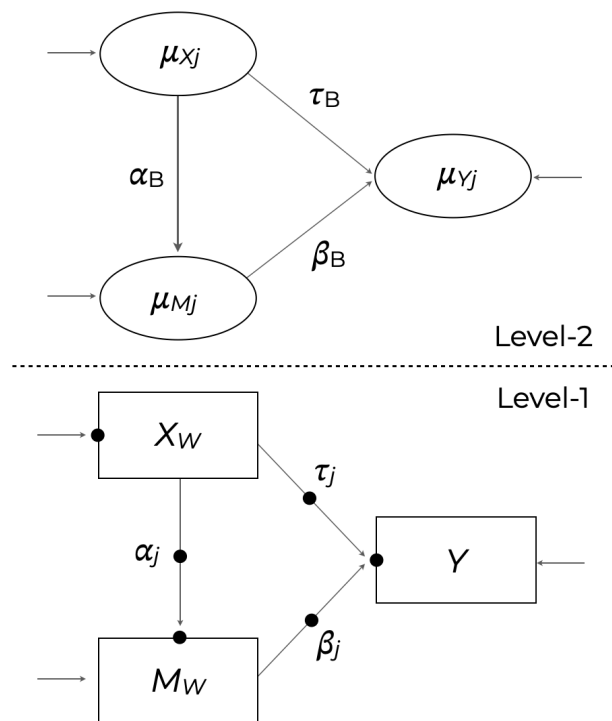
$$\mu_{Y_j} = I_Y + \beta_B(\mu_{M_j}) + \tau'_B(\mu_{X_j}) + u_{Y_j}$$

$$\alpha_j = \mu_\alpha + u_{\alpha_j}$$

$$\beta_j = \mu_\beta + u_{\beta_j}$$

$$\tau'_j = \mu_{\tau'} + u_{\tau'_j}$$

The model features distinct within-cluster and between-cluster mediation processes, as depicted in the path diagram below.



The ellipses in the between-cluster model represent latent group means (i.e., random intercepts). Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.14.imp](#) [data1.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `LATENT` command defines between-cluster latent variables representing the random intercepts and slopes
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ `MODEL` command estimates the random intercept and slope means, using the `->` operator to specify means for all variables that do not require labels
- ❖ `MODEL` command sets the intercept of the regression equation equal to the level-2 latent mean (e.g., `x_i ~ 1@xmean_j`)
- ❖ `MODEL` command centers predictors at their latent group means to obtain pure within-cluster variables (e.g., `x_i - xmean_j`)
- ❖ `MODEL` command omits the random coefficient listed after the vertical pipe
- ❖ `MODEL` command sets the random predictor's slope equal to the random coefficient (e.g., `(x_i - xmean_j)@alpha_j`)
- ❖ `MODEL` command specifies the correlation between random slopes
- ❖ `PARAMETERS` command uses labeled quantities to compute the product of coefficients estimator
- ❖ `PARAMETERS` command uses the `.totalvar` function to access the variance of the random effects

```

DATA: data1.dat;
VARIABLES: level1id level2id v1_i y_i m_i x_i v2_i v3_j v4_j;
CLUSTERID: level2id;
MISSING: 999;
LATENT: level2id = xmean_j mmean_j ymean_j
           alpha_j beta_j tau_j;
MODEL:
  level2.mediation:
  mmean_j ~ xmean_j@alpha_b;
  ymean_j ~ mmean_j@beta_b xmean_j@tau_b;
  level1.mediation:

```

```

m_i ~ 1@mmean_j (x_i - xmean_j)@alpha_j;
y_i ~ 1@ymean_j (m_i - mmean_j)@beta_j (x_i - xmean_j)@tau_j;
level2.models:
1 -> xmean_j mmean_j ymean_j tau_j;
alpha_j ~ 1@alpha_mean;
beta_j ~ 1@beta_mean;
alpha_j ~~ beta_j@ab_cor;
level1.models:
x_i ~ 1@xmean_j;
PARAMETERS:
ab_cov = ab_cor * sqrt(alpha_j.totalvar * beta_j.totalvar);
ab_w = alpha_mean * beta_mean + ab_cov;
ab_b = alpha_b * beta_b;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding rblimp script is as follows.

```

library(rblimp)
load(file = 'data1.rda')

mymodel <- rblimp(
  data = data1,
  clusterid = 'level2id',
  latent = 'level2id = xmean_j mmean_j ymean_j alpha_j beta_j tau_j',
  model = '
level2.mediation:
mmean_j ~ xmean_j@alpha_b;
ymean_j ~ mmean_j@beta_b xmean_j@tau_b;
level1.mediation:
m_i ~ 1@mmean_j (x_i - xmean_j)@alpha_j;
y_i ~ 1@ymean_j (m_i - mmean_j)@beta_j (x_i - xmean_j)@tau_j;
level2.models:
1 -> xmean_j mmean_j ymean_j tau_j;
alpha_j ~ 1@alpha_mean;
beta_j ~ 1@beta_mean;
alpha_j ~~ beta_j@ab_cor;
level1.models:
x_i ~ 1@xmean_j',
  parameters = 'ab_cov = ab_cor * sqrt(alpha_j.totalvar * beta_j.totalvar);
ab_w = alpha_mean * beta_mean + ab_cov;
ab_b = alpha_b * beta_b',

```

```

seed = 90291,
burn = 10000,
iter = 10000
)
output(mymodel)

```

6.15: Two-Level Growth Model

This example illustrates a two-level linear growth model that includes a cross-level group-by-time interaction involving the temporal predictor ($TIME = 0, 1, 3, 6$) and a binary moderator. The regression model, which is the two-level version of the latent growth model from Example 5.17. The multilevel model features a cross-level (group-by-time) interaction effect involving a level-2 dummy code D (e.g., a treatment assignment indicator) and the level-1 time scores, as follows.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j})(TIME_{ij}) + \beta_2(D_j) + \beta_3(TIME_{ij})(D_j) + \varepsilon_{ij}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.15.imp](#) [data9.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `FIXED` command identifies complete predictors
- ❖ `NOMINAL` command identifies a binary predictor
- ❖ `MODEL` command features a random coefficient listed after the vertical pipe
- ❖ `MODEL` command features a product term
- ❖ `SIMPLE` command produces conditional effects (simple slopes) at each level of the nominal moderator


```

DATA: data9.dat;
VARIABLES: level2id y_i time_i v1_i v2_i v3_i d_j v4_j;
CLUSTERID: level2id;
NOMINAL: d_j;
MISSING: 999;
FIXED: time_i d_j;
MODEL: y_i ~ time_i d_j time_i*d_j | time_i;
SIMPLE: time_i | d_j;
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data9.rda')

mymodel <- rblimp(
  data = data9,
  clusterid = 'level2id',
  nominal = 'd_j',
  fixed = 'time_i d_j',
  model = 'y_i ~ time_i d_j time_i*d_j | time_i',
  simple = 'time_i | d_j',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

```

6.16: Three-Level Growth Model

This example illustrates a three-level linear growth model that includes a cross-level group-by-time interaction involving the temporal predictor ($TIME = 0, 1, \dots, 5, 6$) and a level-2 binary moderator. The regression model is as follows.

$$Y_{ijk} = (\beta_0 + b_{0jk} + b_{0k}) + (\beta_1 + b_{1jk} + b_{1k})(TIME_{ijk}) + \beta_2(D_k) + \beta_3(TIME_{ijk})(D_k) + \varepsilon_{ijk}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.16a.imp](#) [Ex6.16b.imp](#) [data10.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies level-2 and level-3 identifiers (order doesn't matter), automatically inducing random intercepts for all level-1 and level-2 variables
- ❖ `FIXED` command identifies complete predictors
- ❖ `CENTER` command applies grand mean centering to predictors
- ❖ `NOMINAL` command identifies a binary predictor
- ❖ `MODEL` command features a random coefficient listed after the vertical pipe
- ❖ `MODEL` command features a product term
- ❖ `SIMPLE` command produces conditional effects (simple slopes) at each level of the nominal moderator

```
DATA: data10.dat;
VARIABLES: level1id level2id level3id y_i time_i v1_i
  v2_j:v5_j v6_j d_k v7_k v8_k;
NOMINAL: d_k;
CLUSTERID: level2id level3id;
MISSING: 999;
FIXED: time_i d_k;
MODEL:
y_i ~ time_i d_k time_i*d_k | time_i;
SIMPLE: time_i | d_k;
SEED: 90291;
BURN: 15000;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data10.rda')

mymodel <- rblimp(
  data = data10,
  nominal = 'd_k',
  clusterid = 'level2id level3id',
  fixed = 'time_i d_k',
  model = 'y_i ~ time_i d_k time_i*d_k | time_i',
  simple = 'time_i | d_k',
  seed = 90291,
  burn = 15000,
  iter = 10000
)
output(mymodel)

```

By default, Blimp estimates random intercepts and random slopes (when specified) at all levels of the data hierarchy. For example, the previous analysis produces a 2 x 2 covariance matrix of random effects at level-2 and level-3. In some situations, it may be desirable or necessary to override Blimp's default behavior and fix certain variance components to zero (or alternatively, select which variances get estimated). This is achieved by listing the desired random effects on the right side of the vertical pipe and appending to the effect's name a cluster-level identifier in square brackets. To illustrate, the following code block illustrates a three-level model with random intercepts at both levels and a random coefficient for the temporal predictor at the second level only.

```

DATA: data10.dat;
VARIABLES: level1id level2id level3id y_i time_i v1_i
  v2_j:v5_j v6_j d_k v7_k v8_k;
NOMINAL: d_k;
CLUSTERID: level2id level3id;
MISSING: 999;
FIXED: time_i d_k;
MODEL:
y_i ~ time_i d_k time_i*d_k |
  1[level2id] 1[level3id] time_i[level2id];
SIMPLE: time_i | d_k;
SEED: 90291;

```

```
BURN: 15000;
ITER: 10000;
```

The corresponding rblimp script is as follows.

```
library(rblimp)
load(file = 'data10.rda')

mymodel <- rblimp(
  data = data10,
  nominal = 'd_k',
  clusterid = 'level2id level3id',
  fixed = 'time_i d_k',
  model = 'y_i ~ time_i d_k time_i*d_k |
    1[level2id] 1[level3id] time_i[level2id]',
  simple = 'time_i | d_k',
  seed = 90291,
  burn = 15000,
  iter = 10000
)
output(mymodel)
```

6.17: Three-Level SEM Growth Model

This example illustrates a three-level linear growth model that includes a cross-level group-by-time interaction involving the temporal predictor ($TIME = 0, 1, \dots, 5, 6$) and a level-2 binary moderator. The regression model from Example 6.16 is as follows.

$$Y_{ijk} = (\beta_0 + b_{0jk} + b_{0k}) + (\beta_1 + b_{1jk} + b_{1k})(TIME_{ijk}) + \beta_2(D_k) + \beta_3(TIME_{ijk})(D_k) + \varepsilon_{ijk}$$

This example casts the analysis as a three-level structural equation model. The level-specific equations are as follows.

$$Y_{ijk} = \beta_{0jk} + \beta_{1jk}(TIME_{ijk}) + \varepsilon_{ijk}$$

$$\beta_{0jk} = \beta_{0k} + b_{0jk}$$

$$\beta_{1jk} = \beta_{1k} + b_{1jk}$$

$$\beta_{0k} = \beta_0 + \beta_2(D_k) + b_{0k}$$

$$\beta_{1k} = \beta_1 + \beta_3(D_k) + b_{1k}$$

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.17.imp](#) [data10.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies level-2 and level-3 identifiers (order doesn't matter), automatically inducing random intercepts for all level-1 and level-2 variables
- ❖ `ORDINAL` command identifies binary predictor
- ❖ `FIXED` command defines complete predictors
- ❖ `CENTER` command applies grand mean and latent group mean centering to predictors
- ❖ `LATENT` command defines two between-cluster latent variables representing the random intercepts and slopes
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ `MODEL` command estimates the random intercept and slope means
- ❖ `MODEL` command sets the intercept of the regression equation equal to the level-2 latent mean (i.e., `1@beta0_j`)
- ❖ `MODEL` command omits the random coefficient listed after the vertical pipe
- ❖ `MODEL` command sets the random predictor's slope equal to the random coefficient (i.e., `x1_i@beta1_j`)
- ❖ `MODEL` command specifies correlation between random intercepts and random slopes (level-2 latent variables)

DATA: data10.dat;

```

VARIABLES: level1id level2id level3id y_i time_i v1_i
  v2_j:v5_j v6_j d_k v7_k v8_k;
NOMINAL: d_k;
CLUSTERID: level2id level3id;
MISSING: 999;
LATENT: level2id = beta0_j beta1_j; level3id = beta0_k beta1_k;
MODEL:
  level3.model:
  beta0_k ~ 1 d_k;
  beta1_k ~ 1 d_k;
  beta0_k ~~ beta1_k;
  level2.model:
  beta0_j ~ 1@beta0_k;
  beta1_j ~ 1@beta1_k;
  beta0_j ~~ beta1_j;
  level1.model:
  y_i ~ 1@beta0_j time_i@beta1_j;
SEED: 90291;
BURN: 30000;
ITER: 30000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data10.rda')

mymodel <- rblimp(
  data = data10,
  nominal = 'd_k',
  clusterid = 'level2id level3id',
  latent = 'level2id = beta0_j beta1_j;
  level3id = beta0_k beta1_k',
  model = '
  level3.model:
  beta0_k ~ 1 d_k;
  beta1_k ~ 1 d_k;
  beta0_k ~~ beta1_k;
  level2.model:
  beta0_j ~ 1@beta0_k;
  beta1_j ~ 1@beta1_k;
  beta0_j ~~ beta1_j;
  level1.model:
  y_i ~ 1@beta0_j time_i@beta1_j',

```

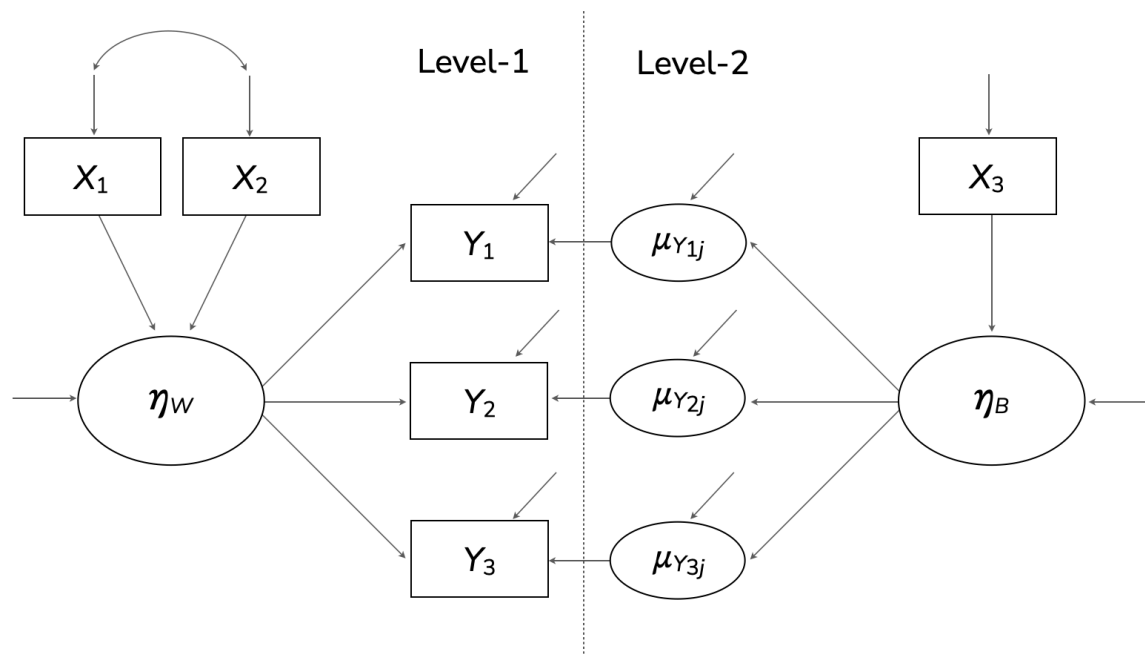
```

seed = 90291,
burn = 30000,
iter = 30000
)
output(mymodel)

```

6.18: Two-Level MIMIC Measurement Model

This example illustrates a two-level factor analysis model that features a measurement model for the within-cluster scores at level-1 and the between-cluster latent group means at level-2. The model also features predictor variables at each level, as shown in the path diagram below.



The ellipses in the between-cluster model are latent group means (i.e., random intercepts that load on a level-2 latent variable). Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.18.imp](#) [data11.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ `LATENT` command defines within- and between-cluster latent variables
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ Individual regression equations specified for each indicator (instead of the `->` convention for latent factors)
- ❖ `MODEL` command fixes the within- and between-cluster loading of the first indicator to one
- ❖ Default specification fixes the latent means equal to zero

```

DATA: data11.dat;
VARIABLES: level2id y1_i:y4_i x1_i x2_i x3_j;
MISSING: 999;
CLUSTERID: level2id;
LATENT: laty_lev1; level2id = laty_lev2;
CENTER: grandmean = x1_i x2_i x3_j;
MODEL:
  structural.model:
  laty_lev1 ~ x1_i x2_i;
  laty_lev2 ~ x3_j;
  measurement.model:
  y1_i ~ laty_lev1@1 laty_lev2@1;
  y2_i ~ laty_lev1 laty_lev2;
  y3_i ~ laty_lev1 laty_lev2;
  y4_i ~ laty_lev1 laty_lev2;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data11.rda')

```



```

mymodel <- rblimp(
  data = data11,
  clusterid = 'level2id',
  latent = 'laty_lev1;
  level2id = 'laty_lev2',
  center = 'grandmean = x1_i x2_i x3_j',
  model = '
  structural.model:
  laty_lev1 ~ x1_i x2_i;
  laty_lev2 ~ x3_j;
  measurement.model:
  y1_i ~ laty_lev1@1 laty_lev2@1;
  y2_i ~ laty_lev1 laty_lev2;
  y3_i ~ laty_lev1 laty_lev2;
  y4_i ~ laty_lev1 laty_lev2',
  seed = 90291,
  burn = 10000,
  iter = 10000
)
output(mymodel)

```

6.19: Sampling Weights

This example illustrates a two-level regression model with random intercepts and sampling (inverse probability) weights at each level. The regression model is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + \beta_1 X_{1ij} + \beta_2 X_{2ij} + \beta_3 X_{3ij} + \varepsilon_{ij}$$

Goldstein (2011, Section 3.4.2) describes MCMC estimation for multilevel models with sampling weights. Level-1 and level-2 sampling weights are rescaled following Goldstein (2011, Section 3.4.1). At level-1, the rescaled weights within a given cluster sum to the cluster size. This is the same as the so-called “cluster” method from Asparouhov (2006). Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.19.imp](#) [data21.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all incomplete level-1 variables
- ❖ **WEIGHTS** command identifies level-1 and level-2 weights (order does not matter)
- ❖ Unspecified associations for predictor variables

```
DATA: data21.dat;  
VARIABLES: level2id level1wgt level2wgt v1 y_i x1_i x2_i x3_i  
             x4_i x5_i;  
CLUSTERID: level2id;  
WEIGHT: level1wgt level2wgt;  
MISSING: 999;  
MODEL: y_i ~ x1_i x2_i x3_i;  
SEED: 90291;  
BURN: 2000;  
ITER: 10000;
```

Note that some data sets may not include level-2 sampling weights, in which case the weight command simplifies as follows.

```
CLUSTERID: level2id;  
WEIGHT: level1wgt;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'data21.rda')  
  
mymodel <- rblimp(  
  data = data21,  
  clusterid = 'level2id',  
  weights = 'level1wgt level2wgt',  
  model = 'y_i ~ x1_i x2_i x3_i',  
  seed = 90291,  
  burn = 2000,  
  iter = 10000  
)  
output(mymodel)
```

6.20: Partially Nested Design (Singleton Clusters)

This example illustrates a two-level regression model from a partially nested design. The example below considers a level-2 binary predictor (e.g., a treatment assignment indicator) where participants in group $D = 1$ (e.g., treatment participants) are clustered in level-2 units but observations in group $D = 0$ are not nested (i.e., are singleton clusters). The regression model below features an interaction between the binary indicator and the random intercept, such that the random effect term drops from the equation if $D = 0$.

$$Y_{ij} = \beta_0 + \beta_1 D_j + D_j b_{0j} + \varepsilon_{ij}$$

More generally, the variable D does not need to have a fixed effect. Outside of an intervention context, D could simply be an indicator that differentiates clustered versus singleton observations (e.g., $D = 0$ is a singleton cluster with a single member, $D = 1$ is an observation that shares cluster membership with other observations). The following random intercept model illustrates this idea.

$$Y_{ij} = \beta_0 + \beta_1 X_{ij} + D_j b_{0j} + \varepsilon_{ij}$$

Blimp estimates this model by defining a level-2 latent variable that interacts with D . Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.20.imp](#) [data19.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all incomplete level-1 variables
- ❖ **LATENT** command defines a level-2 latent variable (random intercept), the mean of which is fixed to zero in the **MODEL** section
- ❖ **MODEL** command eliminates the default random intercept by fixing it to zero (**1@0** after the vertical pipe)

- ❖ MODEL command sets the effect of D equal to the the level-2 random intercept

```

DATA: data19.dat;
VARIABLES: level2id y_i d_j;
MISSING: 999;
CLUSTERID: level2id;
LATENT: level2id = beta0_j;
MODEL:
beta0_j ~ 1@0;
y_i ~ d_j d_j@beta0_j | 1@0;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data19.rda')

mymodel <- rblimp(
  data = data19,
  clusterid = 'level2id',
  latent = 'level2id = beta0_j',
  model = '
beta0_j ~ 1@0;
y_i ~ d_j d_j@beta0_j | 1@0',
  seed = 90291,
  burn = 1000,
  iter = 10000
)
output(mymodel)

```

Alternatively, the model can be fit as a mixed model by fixing the default random intercept variance to zero and adding the multiple membership dummy indicator as a random slope predictor, as follows.

```

DATA: data19.dat;
VARIABLES: level2id y_i d_j;
MISSING: 999;
CLUSTERID: level2id;
MODEL: y_i ~ d_j | 1@0 d_j;

```

```
SEED: 90291;  
BURN: 1000;  
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)  
load(file = 'data19.rda')  
  
mymodel <- rblimp(  
  data = data19,  
  clusterid = 'level2id',  
  model = 'y_i ~ d_j | 1@0 d_j',  
  seed = 90291,  
  burn = 1000,  
  iter = 10000  
)  
output(mymodel)
```

6.21: Discrete-Time Survival Model

This example illustrates a discrete-time survival model using Blimp's multilevel modeling features. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex6.21a.imp](#) [Ex6.21b.imp](#) [data26.dat](#)

The input data set is in stacked (i.e., "person-period") format with each row representing a time interval nested within an individual. The data also include a set of time indicators that dummy code each measurement interval. The example below illustrates a model with six intervals and thus six dummy codes. The outcome variable is an event indicator that equals 0 if the event did not happen in the interval and a 1 if the event did happen in the interval. Figure 11.5 from Singer and Willett (2003) illustrates the data structure.

The basic model is a logistic regression with the binary event indicator regressed on the time dummy codes.

$$\ln \left(\frac{\Pr(\text{Event} = 1)}{1 - \Pr(\text{Event} = 1)} \right) = \alpha_1(t_{1j}) + \alpha_2(t_{2j}) + \alpha_3(t_{3j}) + \alpha_4(t_{4j}) + \alpha_5(t_{5j}) + \alpha_6(t_{6j})$$

Note that the model omits the usual regression intercept. The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates imputed data sets, and adding the `savepredicted` keyword to the `OPTIONS` command saves predicted probabilities (see Example 4.21).

- ❖ `CLUSTERID` command identifies a level-2 identifier, automatically inducing random intercepts for all incomplete level-1 variables
- ❖ `ORDINAL` command identifies a binary outcome
- ❖ `FIXED` command identifies a complete predictor
- ❖ `CENTER` command applies grand mean centering to predictors
- ❖ Applying the `logit` function to the dependent variable on the `MODEL` line requests a logit rather than probit link
- ❖ `MODEL` command eliminates the default fixed and random intercepts by fixing both to zero (the `1@0` after the tilde and vertical pipe)
- ❖ `MODEL` command includes Boolean functions that create time-specific dummy codes (e.g., `time_i==1`)
- ❖ `PARAMETERS` command computes predicted probability of the event at each time point (i.e., hazard probabilities)

```

DATA: data26.dat;
VARIABLES: level2id time_i y_i d_j x_j;
ORDINAL: y_i;
CLUSTERID: level2id;
MISSING: 999;
FIXED: time_i;
MODEL:
logit(y_i) ~ 1@0 (time_i==1)@alpha1 (time_i==2)@alpha2
  (time_i==3)@alpha3 (time_i==4)@alpha4 (time_i==5)@alpha5
  (time_i==6)@alpha6 | 1@0;
PARAMETERS:
hazard.1 = exp(alpha1) / (1 + exp(alpha1));
hazard.2 = exp(alpha2) / (1 + exp(alpha2));
hazard.3 = exp(alpha3) / (1 + exp(alpha3));
hazard.4 = exp(alpha4) / (1 + exp(alpha4));

```

```

hazard.5 = exp(alpha5) / (1 + exp(alpha5));
hazard.6 = exp(alpha6) / (1 + exp(alpha6));
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data26.rda')

mymodel <- rblimp(
  data = data26,
  ordinal = 'y_i',
  clusterid = 'level2id',
  fixed = 'time_i',
  model = '
logit(y_i) ~ 1@0 (time_i==1)@alpha1 (time_i==2)@alpha2 (time_i==3)@alpha3
  (time_i==4)@alpha4 (time_i==5)@alpha5 (time_i==6)@alpha6 | 1@0',
  parameters = 'hazard.1 = exp(alpha1) / (1 + exp(alpha1));
hazard.2 = exp(alpha2) / (1 + exp(alpha2));
hazard.3 = exp(alpha3) / (1 + exp(alpha3));
hazard.4 = exp(alpha4) / (1 + exp(alpha4));
hazard.5 = exp(alpha5) / (1 + exp(alpha5));
hazard.6 = exp(alpha6) / (1 + exp(alpha6))',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

```

The next example expands the model by incorporating a person-level dummy code and continuous covariate as predictors of the hazard function.

$$\ln \left(\frac{\Pr(Event = 1)}{1 - \Pr(Event = 1)} \right) = \alpha_1(t_{1j}) + \alpha_2(t_{2j}) + \alpha_3(t_{3j}) + \alpha_4(t_{4j}) + \alpha_5(t_{5j}) + \alpha_6(t_{6j}) + \beta_1(D_{ij}) + \beta_2(X_{ij}^{cgm})$$

As before, the model omits the usual regression intercept and includes a set of six dummy codes that index the intervals. The code block below is identical to the

previous example, but it defines the binary predictor as ordinal and grand mean centers the continuous covariate.

```

DATA: data26.dat;
VARIABLES: level2id time_i y_i d_j x_j;
ORDINAL: y_i;
CLUSTERID: level2id;
MISSING: 999;
FIXED: time_i;
CENTER: grandmean = x_j;
MODEL:
logit(y_i) ~ 1@0 (time_i==1)@alpha1 (time_i==2)@alpha2
             (time_i==3)@alpha3 (time_i==4)@alpha4 (time_i==5)@alpha5
             (time_i==6)@alpha6 d_j x_j | 1@0;
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data26.rda')

mymodel <- rblimp(
  data = data26,
  ordinal = 'y_i',
  clusterid = 'level2id',
  fixed = 'time_i',
  center = 'grandmean = x_j',
  model = '
logit(y_i) ~ 1@0 (time_i==1)@alpha1 (time_i==2)@alpha2
             (time_i==3)@alpha3 (time_i==4)@alpha4 (time_i==5)@alpha5
             (time_i==6)@alpha6 d_j x_j | 1@0',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

```


7 Missing Not at Random Process Analysis Examples

This section illustrates missing not at random analysis models in Blimp. Following previous chapters, the examples in this section use a generic notation system where variable names usually consist of an alphanumeric prefix and a numeric suffix (e.g., Y , X_1 , X_1N_1 , D_1 , D_2). The letter Y designates a dependent variable, a D prefix denotes a binary dummy variable, an O prefix indicates an ordinal variable, and an N prefix indicates a multicategorical nominal variable. Additionally, the multilevel examples use a “_i” suffix to denote level-1 variables, “_j” for level-2 variables, and “_k” for level-3 variables (e.g., $d_{_j}$ is a level-2 dummy variable, $x_{_i}$ is a continuous predictor measured at level-1). Blimp determines the levels automatically, so the suffixes are meant as a visual aid for understanding the scripts. Finally, the model equations use “cgm” and “cwc” superscripts to indicate grand and group mean centering, respectively. The following list outlines the examples in this section.

- ❖ 7.1: Selection Model for Linear Regression
- ❖ 7.2: Pattern Mixture Model for Linear Regression
- ❖ 7.3: Shared Parameter (Wu–Carroll) Latent Curve Model
- ❖ 7.4: Diggle–Kenward Latent Curve Model
- ❖ 7.5: Factor Analysis With a Selection Model
- ❖ 7.6: Mediation Analysis With a Selection Model
- ❖ 7.7: Two-Level Hedeker-Gibbons Pattern Mixture Growth Model
- ❖ 7.8: Selection Model for a Two-Level Regression With Random Coefficients
- ❖ 7.9: Two-Level Shared Parameter Growth Curve Model

7.1: Selection Model for Linear Regression

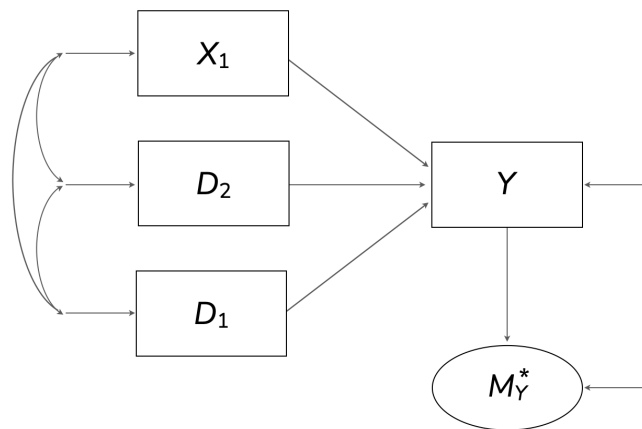
This example illustrates a selection model for a missing not at random process where an incomplete outcome variable predicts its own missingness. The focal analysis model is the linear regression below.

$$Y = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \beta_3 X_1^{cgm} + \varepsilon$$

The most basic selection model is one where the outcome alone predicts its missingness indicator ($M_Y = 0$ if Y is observed and 1 if it's missing); Gomer and Yuan (2021) refer to this as a focused missing not at random process. The following equation is a probit model where the missingness indicator's latent response variable (denoted by an asterisk superscript) is regressed on the outcome.

$$M_Y^* = \gamma_0 + \gamma_1 Y + r$$

For identification, the residual variance is fixed at one, and the threshold parameter is fixed at zero. A path diagram of the model is shown below.



The analysis model also incorporates three auxiliary variables using the sequential specification from Example 4.7. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.1a.imp](#) [Ex7.1b.imp](#) [data3.dat](#)

The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis that no longer requires a missingness model.

- ❖ `ORDINAL` command identifies binary predictors

- ❖ **FIXED** command identifies complete predictors
- ❖ **CENTER** command applies grand mean centering to predictors
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command features a syntax shortcut that creates a factored regression (sequential) specification for auxiliary variables
- ❖ The **.missing** suffix references the dependent variable's missing data indicator, which is automatically defined as ordinal
- ❖ Unspecified associations for predictor variables

```

DATA: data3.dat;
VARIABLES: id x1 x2 x3 y d1 d2 v1:v4;
MISSING: 999;
ORDINAL: d1 d2;
FIXED: d1 d2;
CENTER: x1;
MODEL:
focal.model:
y ~ d1 d2 x1;
missingness.model:
y.missing ~ y;
auxiliary.model:
x2 x3 ~ y d1 d2 x1;
SEED: 90291;
BURN: 1000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data3.rda')

mymodel <- rblimp(
  data = data3,
  ordinal = 'd1 d2',
  fixed = 'd1 d2',
  center = 'x1',
  model = '
  focal.model:

```

```

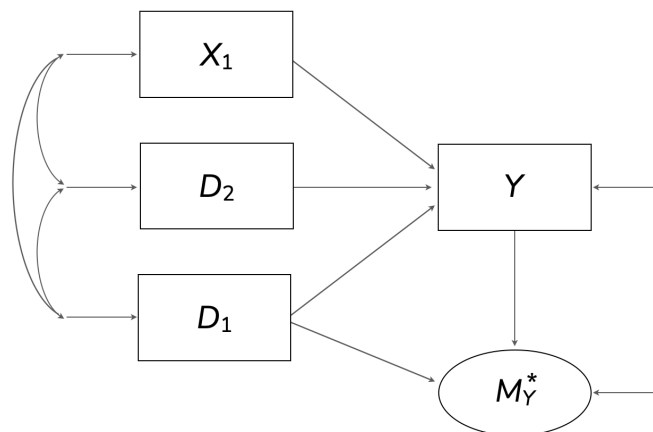
y ~ d1 d2 x1;
missingness.model:
y.missing ~ y;
auxiliary.model:
x2 x3 ~ y d1 d2 x1',
seed = 90291,
burn = 1000,
iter = 10000
)
output(mymodel)

```

A more complex selection model features the outcome predicting its missingness indicator along with other variables, in this case D_1 ; Gomer and Yuan (2021) refer to this as a diffuse missing not at random process. The following equation is a probit model where the missingness indicator's latent response variable is regressed the outcome and D_1 .

$$M_Y^* = \gamma_0 + \gamma_1 Y + \gamma_2 D_1 + r$$

A path diagram of the model is shown below.



Caution is warranted when including too many predictors from the analysis model in the selection equation, as doing so weakens identification. Entering and selecting predictors in a stepwise fashion using fit indices such as the DIC and WAIC is often a good strategy. The code block for the analysis is shown below.

```
DATA: data3.dat;
VARIABLES: id x1 x2 x3 y d1 d2 v1:v4;
MISSING: 999;
ORDINAL: d1 d2;
FIXED: d1 d2;
CENTER: x1;
MODEL:
# focal analysis model
y ~ d1 d2 x1;
# auxiliary variable models
x2 x3 ~ y d1 d2 x1;
# selection model
y.missing ~ y d1;
SEED: 90291;
BURN: 2500;
ITER: 10000;
```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data3.rda')

mymodel <- rblimp(
  data = data3,
  ordinal = 'd1 d2',
  fixed = 'd1 d2',
  center = 'x1',
  model = '
y ~ d1 d2 x1;
x2 x3 ~ y d1 d2 x1;
y.missing ~ y d1',
  seed = 90291,
  burn = 2500,
  iter = 10000
)
output(mymodel)
```

7.2: Pattern Mixture Model for Linear Regression

This example illustrates a pattern mixture model for a missing not at random process where regression model parameters differ between cases with and without dependent variable scores. Clicking the links below downloads the Blimp scripts and

data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.2a.imp](#) [Ex7.2b.imp](#) [data3.dat](#)

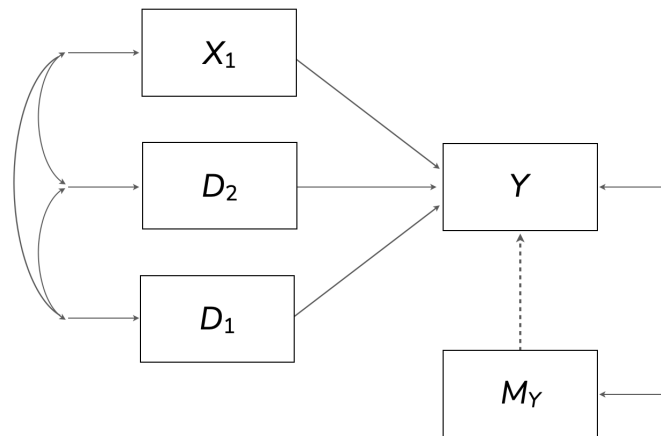
The focal analysis model is the linear regression below

$$Y = \beta_0 + \beta_1 D_1 + \beta_2 D_2^{cgm} + \beta_3 X_1^{cgm} + \varepsilon$$

where D_1 is a dummy code representing a focal group comparison (e.g., a treatment assignment indicator), and D_2 and X_1 are covariates. The most basic pattern mixture model is one where the intercept (outcome variable mean) differs between people with and without Y values; Gomer and Yuan (2021) characterize this as a focused missing not at random process. The fitted model features a binary missing data indicator ($M_Y = 0$ if Y is observed and $M_Y = 1$ if it's missing) as a predictor, as follows.

$$Y = \beta_{0(obs)} + \beta_{0(diff)} M_Y + \beta_1 D_1 + \beta_2 D_2^{cgm} + \beta_3 X_1^{cgm} + \varepsilon$$

A path diagram of the model is shown below.



The overall population-level intercept estimate is a weighted average of the pattern-specific intercepts, where the weights are the group proportions. The marginal intercept estimate for this example is

$$\beta_0 = p_{(obs)}\beta_{0(obs)} + p_{(mis)}\beta_{0(mis)} = p_{(obs)}\beta_{0(obs)} + p_{(mis)}(\beta_{0(obs)} + \beta_{0(diff)})$$

where $p_{(obs)}$ and $p_{(mis)}$ are the proportions of completers and dropouts, respectively.

Importantly, the intercept difference (the dashed line pointing from M_Y to Y) is inestimable because people in the $M_Y = 1$ group have no data on Y . This parameter must be fixed to a value during estimation, and the magnitude and sign of the coefficient controls the strength and direction of the missing not at random process. Enders (2022, Section 9.7) illustrates a strategy that uses off-the-shelf effect size benchmarks to determine this parameter. For example, if a researcher felt that the unseen Y scores have a higher mean than the observed data, then the inestimable intercept coefficient could be solved as a function of the standardized mean difference effect size and the dependent variable's standard deviation (or residual standard deviation).

$$\beta_{0(diff)} = d \times \sigma_Y$$

A positive value of d sets the mean of the unseen scores to a higher value than the observed data, and a negative value specifies a lower mean. The code block below sets the effect size equal to +0.20 and uses the residual standard deviation to estimate the spread of Y (Little, 2009, p. 428). This setting corresponds to a sensitivity analysis where persons with incomplete data are hypothesized to have a mean difference roughly equal to Cohen's (1988) small effect size benchmark. The syntax highlights are listed below. Adding the `NIMPS` and `SAVE` commands generates model-based multiple imputations for a frequentist analysis that no longer requires the missing data indicator.

- ❖ `ORDINAL` command identifies binary predictors
- ❖ `FIXED` command identifies complete predictors
- ❖ `CENTER` command applies grand mean centering to predictors
- ❖ `MODEL` command uses labels ending in a colon to group models and order their summary tables on the output

- ❖ **MODEL** command features a syntax shortcut that creates a factored regression (sequential) specification for all predictor
- ❖ **MODEL** command features a syntax shortcut that creates a factored regression (sequential) specification for auxiliary variables
- ❖ The **TRANSFORM** command creates the dependent variable's missing data indicator, `m.y`
- ❖ **MODEL** command labels the missing data indicator's latent response variable mean and three parameters from the focal analysis model: the residual variance, intercept coefficient, and intercept mean difference
- ❖ **PARAMETERS** command passes the value of the residual standard deviation into the formula that determines the intercept mean difference
- ❖ **PARAMETERS** command uses labeled quantities to compute missing data group proportions, pattern-specific intercept coefficients, and a marginal intercept estimate that averages over the missing data patterns

```

DATA: data3.dat;
VARIABLES: id x1 x2 x3 y d1 d2 v1:v4;
MISSING: 999;
ORDINAL: d1 d2 ymis;
CENTER: x1 d2;
TRANSFORM: ymis = ismissing(y);
MODEL:
focal.model:
y ~ 1@b0obs ymis@b0diff d1 d2 x1;
# label residual variance
y ~~ y@resvar;
predictor.model:
# sequential specification for predictors
ymis ~ 1@ymissmean;
x1 d1 d2 ~ ymis;
auxiliary.model:
x2 x3 ~ y d1 d2 x1;
PARAMETERS:
# set b0diff equal to +.20 residual std. dev. units
cohensd = .20;
b0diff = cohensd * sqrt(resvar);
# missingness group proportions
p_mis = phi(ymissmean);
p_obs = 1 - p_mis;
# compute weighted average intercept

```



```

b0_obs = b0obs;
b0_mis = b0obs + b0diff;
b0 = (b0_obs * p_obs) + (b0_mis * p_mis);
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data3.rda')

mymodel <- rblimp(
  data = data3,
  ordinal = 'd1 d2 ymis',
  center = 'x1 d2',
  transform = 'ymis = ismissing(y)',
  model = '
focal.model:
y ~ 1@b0obs ymis@b0diff d1 d2 x1 ;
y ~~ y@resvar;
predictor.model:
ymis ~ 1@ymissmean;
x1 d1 d2 ~ ymis;
auxiliary.model:
x2 x3 ~ y d1 d2 x1',
  parameters = 'cohensd = .20;
b0diff = cohensd * sqrt(resvar);
p_mis = phi(ymissmean);
p_obs = 1 - p_mis;
b0_obs = b0obs;
b0_mis = b0obs + b0diff;
b0 = (b0_obs * p_obs) + (b0_mis * p_mis)',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

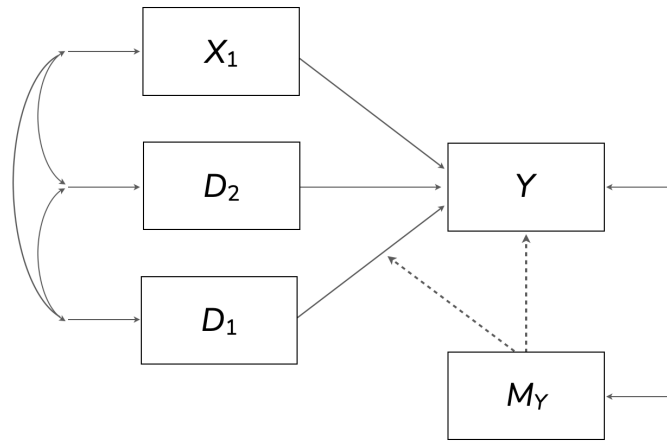
```

A more complex pattern mixture model is one where people with missing outcome scores have different intercepts and slopes than people with data; Gomer and Yuan

(2021) characterize this as a diffuse missing not at random process. The fitted model features the missing data indicator and its interaction with the focal predictor, D_1 .

$$Y = \beta_{0(obs)} + \beta_{0(diff)}M_Y + \beta_{1(obs)}D_1 + \beta_{1(diff)}(D_1 \times M_Y) + \beta_2D_2^{cgm} + \beta_3X_1^{cgm} + \varepsilon$$

A path diagram of the model is as follows.



The marginal slope that averages over the missing data patterns is a weighted average of the pattern-specific slopes, with weights equal to the group proportions.

$$\beta_1 = p_{(obs)}\beta_{1(obs)} + p_{(mis)}\beta_{1(mis)} = p_{(obs)}\beta_{1(obs)} + p_{(mis)}(\beta_{1(obs)} + \beta_{1(diff)})$$

Importantly, both the intercept and slope difference for the incomplete cases (the dashed lines pointing from M_Y to Y and M_Y to D_1 's slope) are inestimable because people in the $M_Y = 1$ group have no data on Y . As such, these parameters must be fixed to a value during estimation, and their magnitude and sign control the strength and direction of the missing not at random process. The same effect size-based strategy can be applied to the slope difference. In this example, the focal predictor D_1 is binary (e.g, intervention vs. control), in which case $\beta_{1(obs)}$ is the group mean difference for people with data on Y , and $\beta_{1(diff)}$ is the additional group mean

difference for persons with missing Y scores. Specifying the inestimable slope as a standardized mean difference effect size gives the following solution.

$$\beta_{1(diff)} = d \times \sigma_Y$$

If the focal predictor is continuous, then the solution is

$$\beta_{1(diff)} = d \times (\sigma_Y \div \sigma_X)$$

in which case d can be viewed as the additional change in the dependent variable (in standard deviation units) for every one standard deviation increase in the predictor. Setting d to a positive value means that the missing data group's slope is more positive, and a negative value of d means their slope is more negative.

To illustrate, suppose that Y is scaled such that high scores reflect a negative outcome (e.g., greater illness severity, a higher symptom count), and D_1 is a treatment assignment dummy code ($D_1 = 0$ indicates the control group, and $D_1 = 1$ is the intervention group). Further, consider a missing not at random process where control group participants with the highest Y scores (e.g., most acute symptoms) leave the study to seek treatment elsewhere, whereas intervention group participants with the lowest Y scores (e.g., mildest symptoms) leave the study because they no longer feel treatment is necessary. This scenario requires a positive value of d for the inestimable intercept difference and a negative value of d for the slope difference. The code block below sets both effect sizes equal to 0.20 (they need not be the same) and uses the residual standard deviation to estimate the spread of Y .

```
DATA: data3.dat;
VARIABLES: id x1 x2 x3 y d1 d2 v1:v4;
MISSING: 999;
ORDINAL: d1 d2 m.y;
TRANSFORM:
m.y = ismissing(y);
CENTER: x1 d2;
MODEL:
```

```

focal.model:
y ~ 1@b0obs m.y@b0diff d1@b1obs d1*m.y@b1diff d2 x1;
# label residual variance
y ~~ y@resvar;
predictor.model:
m.y ~ 1@ymissmean;
x1 d1 d2 ~ m.y;
auxiliary.model:
x2 x3 ~ y x1 d1 d2;
PARAMETERS:
# set b0diff equal to +.20 residual std. dev. units
# set b1diff equal to -.20 residual std. dev. units
cohensd = .20;
b0diff = cohensd * sqrt(resvar);
b1diff = - cohensd * sqrt(resvar);
# missingness group proportions
p.mis = phi(ymissmean);
p.obs = 1 - p.mis;
# compute weighted average intercept and slope
b0.obs = b0obs;
b0.mis = b0obs + b0diff;
b1.obs = b1obs;
b1.mis = b1obs + b1diff;
b0 = (b0.obs * p.obs) + (b0.mis * p.mis);
b1 = (b1.obs * p.obs) + (b1.mis * p.mis);
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data3.rda')

mymodel <- rblimp(
  data = data3,
  ordinal = 'd1 d2 m.y',
  transform = 'm.y = ismissing(y)',
  center = 'x1 d2',
  model = '
focal.model:
y ~ 1@b0obs m.y@b0diff d1@b1obs d1*m.y@b1diff d2 x1;

```

```

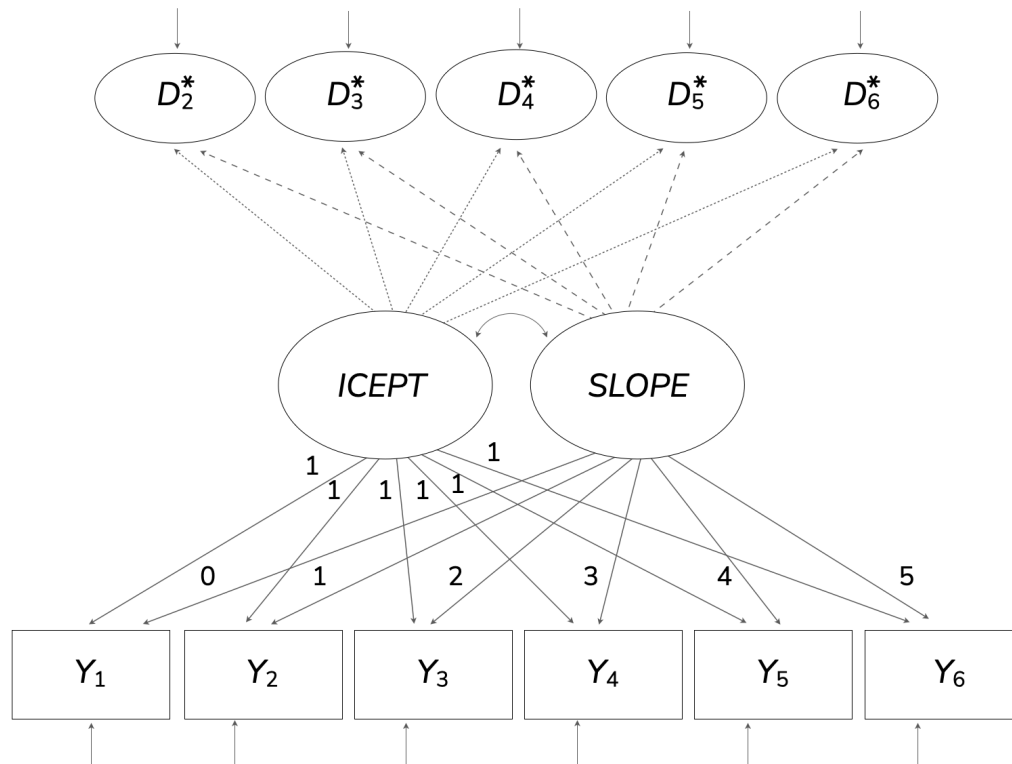
y ~~ y@resvar;
predictor.model:
m.y ~ 1@ymissmean;
x1 d1 d2 ~ m.y;
auxiliary.model:
x2 x3 ~ y x1 d1 d2',
parameters = 'cohensd = .20;
b0diff = cohensd * sqrt(resvar);
b1diff = - cohensd * sqrt(resvar);
p.mis = phi(ymissmean);
p.obs = 1 - p.mis ;
b0.obs = b0obs;
b0.mis = b0obs + b0diff;
b1.obs = b1obs;
b1.mis = b1obs + b1diff;
b0 = (b0.obs * p.obs) + (b0.mis * p.mis);
b1 = (b1.obs * p.obs) + (b1.mis * p.mis)',
seed = 90291,
burn = 2000,
iter = 10000
)
output(mymodel)

```

Linking inestimable parameters to the standardized mean difference provides a practical heuristic for specifying inestimable coefficients, but it is still incumbent on the researcher to choose values that are reasonable for a given application. As mentioned previously, the magnitude of the missing data group's difference parameters dictates the strength of the missing not at random process. It is incorrect to view "small" values of d as unimportant, as standardized differences of this magnitude could be very salient in many situations. For example, consider a randomized intervention where the true effect size is $d = 0.20$ (i.e., a small effect size). Setting the missing data group's coefficient difference to $d = .20$ means that the moderating impact of missing data is just as large as the intervention effect itself. A medium effect size threshold is probably an upper bound for most practical applications, and much smaller values of d could be realistic. A sensitivity analysis strategy would examine the changes in the focal model parameters across a range of d values (see Enders 2022, Section 9.8).

7.3: Shared Parameter (Wu–Carroll) Latent Curve Model

This example illustrates a two-factor latent growth curve model with the Wu–Carroll (1998) model for missing not at random dropout that depends on the growth trajectories. A path diagram of the model is shown below.



The model features binary dropout indicators regressed on the random intercepts and slopes. Importantly, the missing data indicators code attrition or permanent dropout rather than intermittent missingness. Accordingly, the D variables equal 0 prior to dropout, 1 at the occasion participants leave the study, and 999 (missing) at all post-dropout measurements. There is no missing data indicator for Y_1 because this variable is complete. The path coefficients connecting dropout at occasion t to the random intercepts are constrained to equality, as are the coefficients connecting the random slopes to dropout. The dashed lines convey these constraints. Predictor variables can be incorporated into either the outcome or dropout part of the model.

Additional details about this model can be found in Enders (2011) and Muthén, Asparouhov, Hunter, and Leuchter (2011).

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.3.imp](#) [data24.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies the binary dropout indicators
- ❖ **LATENT** command defines two latent variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ Individual regression equations specified for each indicator (instead of the -> convention for latent factors)
- ❖ **MODEL** command estimates the latent variable means, fixes the intercept factor loadings to one, fixes the growth factor loadings to the time scores (0, 1, 2, 3, 4, and 5), and fixes the measurement intercepts to zero
- ❖ **MODEL** command uses a label to impose equality constraint on residual variance, a label to constrain associations between the random intercepts and indicators, and a label to constrain associations between the random slopes and indicators

```

DATA: data24.dat;
VARIABLES: id y1 y2 y3 y4 y5 y6 d1 d2 d3 d4 d5 d6;
ORDINAL: d2 d3 d4 d5 d6;
MISSING: 999;
LATENT: icept slope;
MODEL:
  structural.model:
  icept ~ 1;
  slope ~ 1;
  icept ~~ slope;
  measurement.model:
  y1 ~ 1@0 icept@1 slope@0;
  y2 ~ 1@0 icept@1 slope@1;

```

```

y3 ~ 1@0 icept@1 slope@2;
y4 ~ 1@0 icept@1 slope@3;
y5 ~ 1@0 icept@1 slope@4;
y6 ~ 1@0 icept@1 slope@5;
y1 ~~ y1@vconstraint;
y2 ~~ y2@vconstraint;
y3 ~~ y3@vconstraint;
y4 ~~ y4@vconstraint;
y5 ~~ y5@vconstraint;
y6 ~~ y6@vconstraint;
dropout.model:
d2 ~ icept@iconstraint slope@sconstraint;
d3 ~ icept@iconstraint slope@sconstraint;
d4 ~ icept@iconstraint slope@sconstraint;
d5 ~ icept@iconstraint slope@sconstraint;
d6 ~ icept@iconstraint slope@sconstraint;
SEED: 90291;
BURN: 100000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data24.rda')

mymodel <- rblimp(
  data = data24,
  ordinal = 'd2 d3 d4 d5 d6',
  latent = 'icept slope',
  model = '
  structural.model:
  icept ~ 1;
  slope ~ 1;
  icept <-> slope;
  measurement.model:
  y1 ~ 1@0 icept@1 slope@0;
  y2 ~ 1@0 icept@1 slope@1;
  y3 ~ 1@0 icept@1 slope@2;
  y4 ~ 1@0 icept@1 slope@3;
  y5 ~ 1@0 icept@1 slope@4;
  y6 ~ 1@0 icept@1 slope@5;

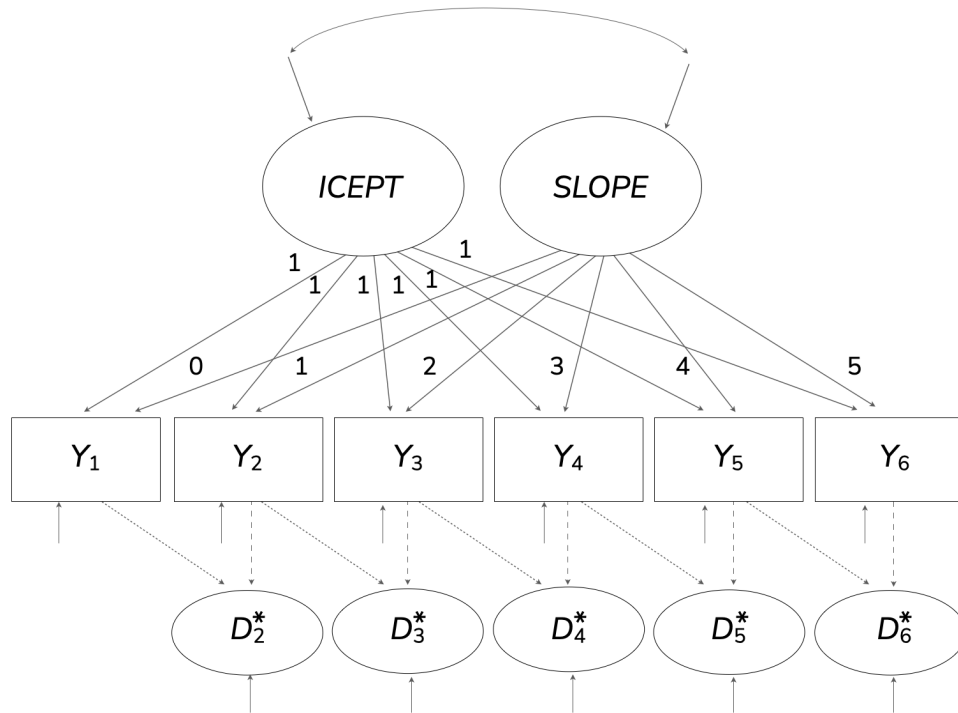
```



```
y1 ~~ y1@vconstraint;  
y2 ~~ y2@vconstraint;  
y3 ~~ y3@vconstraint;  
y4 ~~ y4@vconstraint;  
y5 ~~ y5@vconstraint;  
y6 ~~ y6@vconstraint;  
dropout.model:  
d2 ~ icept@iconstraint slope@sconstraint;  
d3 ~ icept@iconstraint slope@sconstraint;  
d4 ~ icept@iconstraint slope@sconstraint;  
d5 ~ icept@iconstraint slope@sconstraint;  
d6 ~ icept@iconstraint slope@sconstraint',  
seed = 90291,  
burn = 100000,  
iter = 10000  
)  
output(mymodel)
```

7.4: Diggle-Kenward Latent Curve Model

This example illustrates a two-factor latent growth curve model with the Diggle–Kenward (1998) model for missing not at random dropout that depends on the unseen outcome score at time t . A path diagram of the model is shown below.



The model features binary dropout indicators regressed on the random intercepts and slopes. Importantly, the missing data indicators code attrition or permanent dropout rather than intermittent missingness. Accordingly, the D variables equal 0 prior to dropout, 1 at the occasion participants leave the study, and 999 (missing) at all post-dropout measurements. There is no missing data indicator for Y_1 because this variable is complete. The path coefficients connecting dropout at occasion t to the observed data from the previous occasion are constrained to equality, as are the paths connecting dropout at occasion t with the concurrent (unseen) scores at occasion t . The dashed lines convey these constraints. Predictor variables can be incorporated into either the outcome or dropout part of the model. Additional details about this model can be found in Enders (2011) and Muthén, Asparouhov, Hunter, and Leuchter (2011).

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.4.imp](#) [data23.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies the binary dropout indicators
- ❖ **LATENT** command defines two latent variables
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ Individual regression equations specified for each indicator (instead of the -> convention for latent factors)
- ❖ **MODEL** command estimates the latent variable means, fixes the intercept factor loadings to one, fixes the growth factor loadings to the time scores (0, 1, 2, 3, 4, and 5), and fixes the measurement intercepts to zero
- ❖ **MODEL** command uses a label to impose equality constraint on residual variance, a label to constrain lagged associations between the outcomes and indicators, and a label to constrain concurrent associations between the outcomes and indicators
- ❖ Longer burn-in period for estimating latent variables

```

DATA: data23.dat;
VARIABLES: id y1 y2 y3 y4 y5 y6 d1 d2 d3 d4 d5 d6;
ORDINAL: d2 d3 d4 d5 d6;
MISSING: 999;
LATENT: icept slope;
MODEL:
  structural.model:
  icept ~ 1;
  slope ~ 1;
  icept ~~ slope;
  measurement.model:
  y1 ~ 1@0 icept@1 slope@0;
  y2 ~ 1@0 icept@1 slope@1;
  y3 ~ 1@0 icept@1 slope@2;
  y4 ~ 1@0 icept@1 slope@3;
  y5 ~ 1@0 icept@1 slope@4;
  y6 ~ 1@0 icept@1 slope@5;
  y1 ~~ y1@vconstraint;

```

```

y2 ~~ y2@vconstraint;
y3 ~~ y3@vconstraint;
y4 ~~ y4@vconstraint;
y5 ~~ y5@vconstraint;
y6 ~~ y6@vconstraint;
dropout.model:
d2 ~ y1@marconstraint y2@mnarconstraint;
d3 ~ y2@marconstraint y3@mnarconstraint;
d4 ~ y3@marconstraint y4@mnarconstraint;
d5 ~ y4@marconstraint y5@mnarconstraint;
d6 ~ y5@marconstraint y6@mnarconstraint;
SEED: 90291;
BURN: 100000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data23.rda')

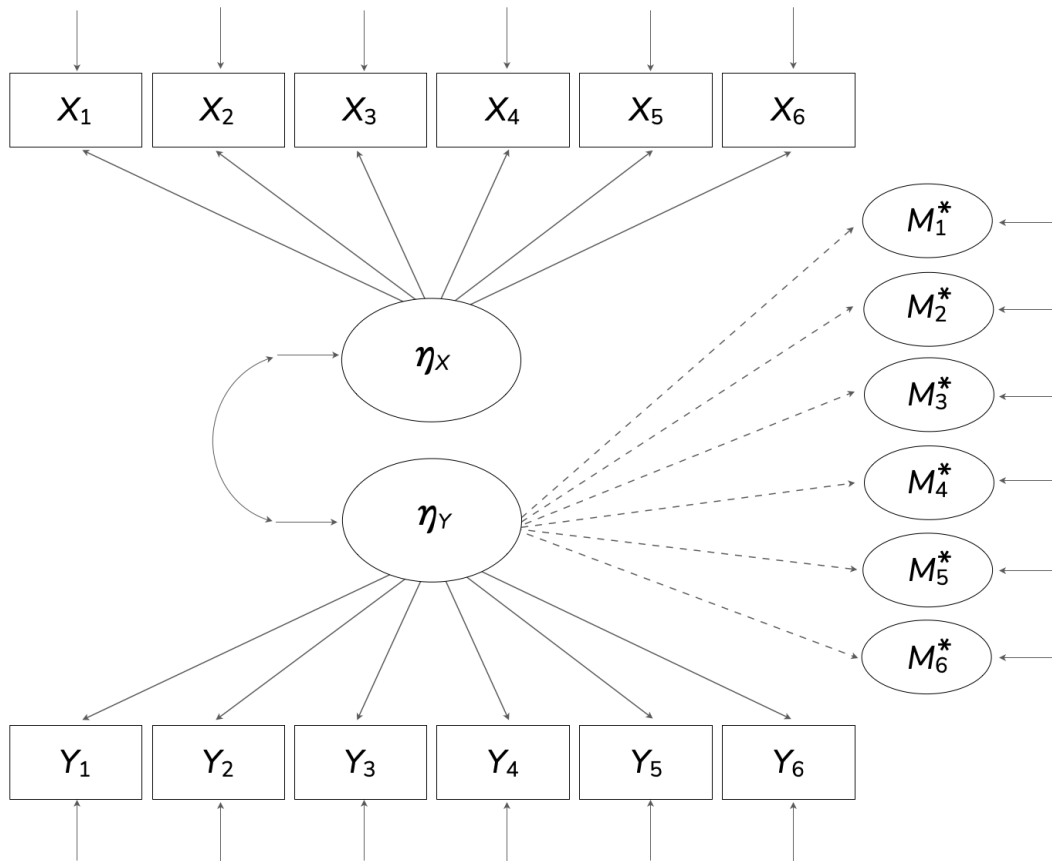
mymodel <- rblimp(
  data = data23,
  ordinal = 'd2 d3 d4 d5 d6',
  latent = 'icept slope',
  model = '
structural.model:
icept ~ 1;
slope ~ 1;
icept <-> slope;
measurement.model:
y1 ~ 1@0 icept@1 slope@0;
y2 ~ 1@0 icept@1 slope@1;
y3 ~ 1@0 icept@1 slope@2;
y4 ~ 1@0 icept@1 slope@3;
y5 ~ 1@0 icept@1 slope@4;
y6 ~ 1@0 icept@1 slope@5;
y1 ~~ y1@vconstraint;
y2 ~~ y2@vconstraint;
y3 ~~ y3@vconstraint;
y4 ~~ y4@vconstraint;
y5 ~~ y5@vconstraint;
y6 ~~ y6@vconstraint;

```

```
dropout.model:  
d2 ~ y1@marconstraint y2@mnarconstraint;  
d3 ~ y2@marconstraint y3@mnarconstraint;  
d4 ~ y3@marconstraint y4@mnarconstraint;  
d5 ~ y4@marconstraint y5@mnarconstraint;  
d6 ~ y5@marconstraint y6@mnarconstraint',  
seed = 90291,  
burn = 100000,  
iter = 10000  
)  
output(mymodel)
```

7.5: Factor Analysis With a Selection Model

This example illustrates a two-factor measurement model with correlated latent variables, each measured by six continuous indicators. A path diagram of the analysis model is shown in Section 5.5. The main difference is that the analysis incorporates a selection missingness model for the indicators of one latent factor. The selection equations model a missing not at random process where missingness is explained by an individual's standing on the latent trait. A path diagram of the model is shown below.



The M variables are binary missing data indicators coded 0 if a response is complete and 1 if missing. The path coefficients connecting the missingness indicators to the latent factor are constrained to equality (i.e., one's standing on the factor exerts the same influence on the probability of missing not at random missingness). The dashed lines convey these constraints.

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.5.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ LATENT command defines two latent variables

- ❖ TRANSFORM command uses the `ismissing` function to create missing data indicators for the six Y items
- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command uses a label to impose equality constraint on the associations between the latent factor and the missingness indicators
- ❖ MODEL command fixes variances and residual variances to one for identification
- ❖ PARAMETERS command specifies a truncated prior over positive values, and the prior is attached to each factor's first loading in the MODEL command

```

DATA: data4.dat;
VARIABLES: id v1:v9 y1:y6 v10:v16 x1:x6;
MISSING: 999;
ORDINAL: y1mis y2mis y3mis y4mis y5mis y6mis;
TRANSFORM:
y1mis = ismissing(y1);
y2mis = ismissing(y2);
y3mis = ismissing(y3);
y4mis = ismissing(y4);
y5mis = ismissing(y5);
y6mis = ismissing(y6);
LATENT: latenty latentx;
MODEL:
latent.model:
latentx ~~ latentx@1;
latenty ~~ latenty@1;
latentx ~~ latenty;
measurement.models:
latentx -> x1@xload_prior x2:x6;
latenty -> y1@yload_prior y2:y6;
missingness.model:
y1mis ~ latenty@misconstraint;
y2mis ~ latenty@misconstraint;
y3mis ~ latenty@misconstraint;
y4mis ~ latenty@misconstraint;
y5mis ~ latenty@misconstraint;
y6mis ~ latenty@misconstraint;
PARAMETERS:
xload_prior ~ truncate(0, Inf);
yload_prior ~ truncate(0, Inf);
SEED: 90291;

```

```
BURN: 20000;  
ITER: 20000;
```

The corresponding rblimp script is as follows.

```
library(rblimp)  
load(file = 'data4.rda')  
  
mymodel <- rblimp(  
  data = data4,  
  ordinal = 'y1mis y2mis y3mis y4mis y5mis y6mis',  
  transform = 'y1mis = ismissing(y1);  
y2mis = ismissing(y2);  
y3mis = ismissing(y3);  
y4mis = ismissing(y4);  
y5mis = ismissing(y5);  
y6mis = ismissing(y6)',  
  latent = 'latency latentx',  
  model = '  
latent.model:  
latentx ~ latentx@1;  
latency ~ latency@1;  
latentx ~ latency;  
measurement.models:  
latentx -> x1@xload_prior x2:x6;  
latency -> y1@yload_prior y2:y6;  
missingness.model:  
y1mis ~ latency@misconstraint;  
y2mis ~ latency@misconstraint;  
y3mis ~ latency@misconstraint;  
y4mis ~ latency@misconstraint;  
y5mis ~ latency@misconstraint;  
y6mis ~ latency@misconstraint',  
  parameters = 'xload_prior ~ truncate(0,Inf);  
yload_prior ~ truncate(0,Inf)',  
  seed = 90291,  
  burn = 20000,  
  iter = 20000  
)  
output(mymodel)
```


7.6: Mediation Analysis With a Selection Model

This example illustrates a single-mediator path model with selection models for a missing not at random process on the mediator and outcome. The focal regression models are shown below

$$M = I_M + \alpha X + \varepsilon_M$$

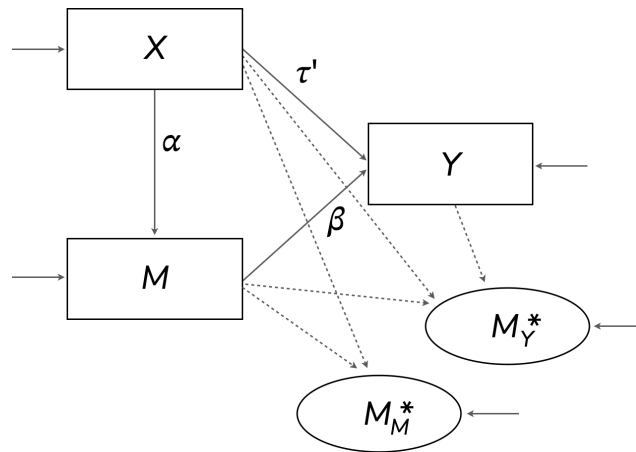
$$Y = I_Y + \beta M + \tau' X + \varepsilon_Y$$

where α and β are slope coefficients that define the indirect effect or product of the coefficients estimator, and τ' is the direct effect of X on Y . A path diagram of the analysis is shown in Section 5.1. The analysis additionally incorporates missingness models for both M and Y .

$$M_{mis}^* = \gamma_{01} + \gamma_{11}M + \gamma_{21}X + \varepsilon_1$$

$$Y_{mis}^* = \gamma_{02} + \gamma_{12}Y + \gamma_{22}M + \gamma_{32}X + \varepsilon_2$$

The asterisk superscripts on the binary missing data indicators represent latent response variables from a probit regression. M 's missingness model features M and X as predictors, and Y 's missingness model features Y , M , and X . Finally, the model also incorporates three auxiliary variables following the procedure from Example 4.7. A path diagram of the focal model is shown below.



Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.6.imp](#) [data4.dat](#)

The syntax highlights are as follows.

- ❖ **TRANSFORM** command uses the `ismissing` function to create missing data indicators for M and Y
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command labels the indirect effect's component pathways
- ❖ **MODEL** command features a syntax shortcut that creates a factored regression (sequential) specification for auxiliary variables
- ❖ **PARAMETERS** command uses labeled quantities to compute the product of coefficients estimator

```

DATA: data4.dat;
VARIABLES: id a1:a3 v1 y m v2 x v3:v22;
MISSING: 999;
TRANSFORM:
m.mis = ismissing(m);

```

```

y.mis = ismissing(y);
MODEL:
mediation.model:
m ~ x@alpha;
y ~ m@beta x;
missingness.model:
m.mis ~ m x;
y.mis ~ y m x;
auxiliary.model:
# sequential specification for auxiliary variables
a1:a3 ~ y m x;
PARAMETERS:
indirect = alpha * beta;
SEED: 90291;
BURN: 2000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data4.rda')

mymodel <- rblimp(
  data = data4,
  transform = 'm.mis = ismissing(m);
y.mis = ismissing(y)',
  model = '
mediation.model:
m ~ x@alpha;
y ~ m@beta x;
missingness.model:
m.mis ~ m x;
y.mis ~ y m x;
auxiliary.model:
a1:a3 ~ y m x',
  parameters = 'indirect = alpha * beta',
  seed = 90291,
  burn = 2000,
  iter = 10000
)
output(mymodel)

```

7.7: Two-Level Hedeker-Gibbons Pattern Mixture Growth Model

This example illustrates the random coefficient pattern mixture model from Hedeker and Gibbons (1997). The model is designed for a missing not at random process where growth model parameters differ between cases who complete the study versus those who dropout. Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.7.imp](#) [data18.dat](#)

The multilevel model features a cross-level (group-by-time) interaction effect involving a level-2 dummy code D (e.g., a treatment assignment indicator) and the level-1 time scores, as follows.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j})(TIME_{ij}) + \beta_2(D_j) + \beta_3(TIME_{ij})(D_j) + \varepsilon_{ij}$$

The pattern mixture model introduces a dropout indicator that differentiates completers and dropouts, $M = 0$ and 1 , respectively. The fitted model features the dropout indicator and its interaction effects

$$\begin{aligned} Y_{ij} = & \beta_{0(obs)} + \beta_{1(obs)}TIME_{ij} + \beta_{2(obs)}(D_j) + \beta_{3(obs)}(TIME_{ij})(D_j) \\ & + \beta_{0(diff)}(M_j) + \beta_{1(diff)}(TIME_{ij})(M_j) + \beta_{2(diff)}(D_j)(M_j) \\ & + \beta_{3(diff)}(TIME_{ij})(D_j)(M_j) + b_{0j} + b_{1j}(TIME_{ij}) + \varepsilon_{ij} \end{aligned}$$

where the “obs” subscript denotes the completer group’s ($M = 0$) parameters, and the “diff” subscript denotes coefficient differences for the dropout group ($M = 1$). Following Example 7.2, the overall population-level estimates (i.e., the marginal estimates that average over the distribution of missingness) are a weighted average of the pattern-specific coefficients, where the weights are the group proportions. The marginal estimates for this example are shown below, where $p_{(obs)}$ and $p_{(mis)}$ are the proportions of completers and dropouts, respectively.

$$\beta_0 = p_{(obs)}\beta_{0(obs)} + p_{(mis)}\beta_{0(mis)} = p_{(obs)}\beta_{0(obs)} + p_{(mis)}(\beta_{0(obs)} + \beta_{0(diff)})$$

$$\beta_1 = p_{(obs)}\beta_{1(obs)} + p_{(mis)}\beta_{1(mis)} = p_{(obs)}\beta_{1(obs)} + p_{(mis)}(\beta_{1(obs)} + \beta_{1(diff)})$$

$$\beta_2 = p_{(obs)}\beta_{2(obs)} + p_{(mis)}\beta_{2(mis)} = p_{(obs)}\beta_{2(obs)} + p_{(mis)}(\beta_{2(obs)} + \beta_{2(diff)})$$

$$\beta_3 = p_{(obs)}\beta_{3(obs)} + p_{(mis)}\beta_{3(mis)} = p_{(obs)}\beta_{3(obs)} + p_{(mis)}(\beta_{3(obs)} + \beta_{3(diff)})$$

The syntax highlights are listed below. Adding the **NIMPS** and **SAVE** commands generates model-based multiple imputations for a frequentist analysis (see Example 6.3).

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ **ORDINAL** command identifies binary predictors
- ❖ **CENTER** command applies grand mean and latent group mean centering to predictors
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ **MODEL** command features a random coefficient listed after the vertical pipe
- ❖ **MODEL** command labels each fixed effect coefficient
- ❖ **MODEL** command features a factored regression (sequential) specification for the binary predictors
- ❖ **PARAMETERS** command uses labeled quantities to compute population-average (marginal) coefficients that average over missing data patterns

```

DATA: data18.dat;
VARIABLES: level2id v1_j d_j y_i time_i v2_j m_j v3_i;
ORDINAL: d_j m_j;
CLUSTERID: level2id;
MISSING: 999;
FIXED: time_i;
MODEL:
focal.model:
y_i ~ 1@beta0_obs time_i@beta1_obs d_j@beta2_obs
(time_i*d_j)@beta3_obs m_j@beta0_dif (m_j*time_i)@beta1_dif
(m_j*d_j)@beta2_dif (m_j*time_i*d_j)@beta3_dif | time_i;

```

```

predictor.model:
m_j ~ 1@ymissmean;
d_j ~ m_j;
PARAMETERS:
p_mis = phi(ymissmean);
p_obs = 1 - p_mis ;
beta0 = p_obs * beta0_obs + p_mis * (beta0_obs + beta0_dif);
beta1 = p_obs * beta1_obs + p_mis * (beta1_obs + beta1_dif);
beta2 = p_obs * beta2_obs + p_mis * (beta2_obs + beta2_dif);
beta3 = p_obs * beta3_obs + p_mis * (beta3_obs + beta3_dif);
SEED: 90291;
BURN: 5000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data18.rda')

mymodel <- rblimp(
  data = data18,
  ordinal = 'd_j m_j',
  clusterid = 'level2id',
  fixed = 'time_i',
  model = '
focal.model:
y_i ~ 1@beta0_obs time_i@beta1_obs d_j@beta2_obs
(time_i*d_j)@beta3_obs m_j@beta0_dif (m_j*time_i)@beta1_dif
(m_j*d_j)@beta2_dif (m_j*time_i*d_j)@beta3_dif | time_i;
predictor.model:
m_j ~ 1@ymissmean;
d_j ~ m_j',
  parameters = 'p_mis = phi(ymissmean);
p_obs = 1 - p_mis ;
beta0 = p_obs * beta0_obs + p_mis * (beta0_obs + beta0_dif);
beta1 = p_obs * beta1_obs + p_mis * (beta1_obs + beta1_dif);
beta2 = p_obs * beta2_obs + p_mis * (beta2_obs + beta2_dif);
beta3 = p_obs * beta3_obs + p_mis * (beta3_obs + beta3_dif)',
  seed = 90291,
  burn = 5000,
  iter = 10000
)
output(mymodel)

```

7.8: Selection Model for a Two-Level Regression With Random Coefficients

This example illustrates a two-level regression model with random intercepts and random slopes. The analysis model is shown below.

$$Y_{ij} = (\beta_0 + b_{0j}) + (\beta_1 + b_{1j}) X_{1ij}^{cwc} + \beta_2 X_{2ij}^{cgm} + \beta_3 X_{3j}^{cgm} + \beta_4 D_j^{cgm} + \varepsilon_{ij}$$

The analysis additionally incorporates a missingness model for Y .

$$M_{Yij}^* = \gamma_0 + \gamma_1 Y_{ij} + \gamma_2 X_{1ij}^{cwc} + \varepsilon_{1ij}$$

The asterisk superscript on the binary missing data indicator represents a latent response variable from a probit regression. Y 's missingness model features Y and X_1 as predictors. The M variable is a binary missing data indicator coded 0 if Y is observed and 1 if it is missing. This model would be appropriate for a multilevel model that does not involve a time trend and permanent attrition (e.g., intensive longitudinal data).

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.8.imp](#) [data8.dat](#)

The syntax highlights are as follows.

- ❖ **CLUSTERID** command identifies a level-2 identifier, automatically inducing random intercepts for all level-1 variables
- ❖ **ORDINAL** command identifies a binary predictor
- ❖ **TRANSFORM** command uses the `ismissing` function to create a missing data indicator for Y
- ❖ **FIXED** command identifies a complete predictor
- ❖ **CENTER** command applies grand mean and latent group mean centering to predictors
- ❖ **MODEL** command uses labels ending in a colon to group models and order their summary tables on the output

- ❖ MODEL command features a random coefficient listed after the vertical pipe
- ❖ Unspecified associations for predictor variables

```

DATA: data8.dat;
VARIABLES: level1id level2id x1_i x2_i y_i v1_i v2_i d_j
  v3_j v4_j v5_j x3_j v6_j v7_j;
CLUSTERID: level2id;
ORDINAL: d_j ymis_i;;
MISSING: 999;
TRANSFORM: ymis_i = ismissing(y_i);
FIXED: d_j;
CENTER: groupmean = x1_i; grandmean = x2_i x3_j d_j;
MODEL:
focal.model:
y_i ~ x1_i x2_i x3_j d_j | x1_i;
missingness.model:
ymis_i ~ y_i x1_i;
SEED: 90291;
BURN: 10000;
ITER: 20000;

```

The corresponding `rblimp` script is as follows.

```

library(rblimp)
load(file = 'data8.rda')

mymodel <- rblimp(
  data = data8,
  clusterid = 'level2id',
  ordinal = 'd_j ymis_i;',
  transform = 'ymis_i = ismissing(y_i)',
  fixed = 'd_j',
  center = 'groupmean = x1_i;
  grandmean = x2_i x3_j d_j',
  model = '
  focal.model:
  y_i ~ x1_i x2_i x3_j d_j | x1_i;
  missingness.model:
  ymis_i ~ y_i x1_i',
  seed = 90291,
  burn = 10000,

```



```

    iter = 20000
  )
output(mymodel)

```

7.9: Two-Level Shared Parameter (Wu–Carroll) Growth Curve Model

This example illustrates how to estimate the Wu–Carroll (1998) model from Section 7.3 as a multilevel regression model. Consistent with that earlier example, the model is for situations where missing not at random dropout depends on the growth trajectories. The model features binary dropout indicators regressed on the random intercepts and slopes. Importantly, the missing data indicators code attrition or permanent dropout rather than intermittent missingness. Accordingly, the D variables equal 0 prior to dropout, 1 at the occasion participants leave the study, and 999 (missing) at all post-dropout measurements. In the multilevel specification, the dropout indicators are stacked in a single column like other time-varying variables.

The growth curve model is cast as a multilevel structural equation model with a pair of normally distributed level-2 latent variables representing the random intercepts and slopes. The focal model features these latent variables as predictors, as shown below.

$$Y_{ij} = \beta_{0j} + \beta_{1j}(TIME_{ij}) + \varepsilon_{ij}$$

$$\beta_{0j} = \beta_0 + b_{0j}$$

$$\beta_{1j} = \beta_1 + b_{1j}$$

The latent curve model from the earlier example has features that require special attention when specifying the analysis as a multilevel model. First, there is no dropout indicator for the baseline assessment because this variable is complete. In the multilevel framework, the indicator takes on constant values of zero in the first row of each individual's missingness vector. Second, the random intercepts and slopes cannot influence the omitted (and constant) baseline missingness indicator. Third, each measurement occasion has a unique intercept that determines the

occasion-specific missingness rate. The multilevel selection equation honors these important features.

The multilevel missingness model is shown below.

$$M_{Yij}^* = \gamma_1 T_{1j} + \gamma_2 T_{2j} + \gamma_3 T_{3j} + \gamma_4 T_{4j} + \gamma_5 T_{5j} + \gamma_6 T_{6j} \\ + \gamma_7 B_{ij} \beta_{0j} + \gamma_8 B_{ij} \beta_{1j} + r_{ij}$$

To accommodate occasion-specific intercepts, the missingness model includes a dummy code for each measurement occasion. The equation denotes the time codes as T_1 through T_6 . Note that the usual regression intercept is omitted from the equation (alternatively, the intercept could be estimated if T_1 is omitted). Importantly, the coefficient for the first (baseline) dummy code is fixed at $\gamma_1 = -3$ to induce a near-zero probability of missingness at baseline. In the latent curve model, this is equivalent to omitting the first dropout indicator. Next, the variable B_{ij} is a dummy code that equals 0 at the baseline assessment and 1 in all other rows. Including the product of this dummy code and the random effects acts like an on/off switch that sets the influence of the random intercepts and slopes to zero at baseline. Recall that this was an important feature of the latent curve model. The Blimp script below uses Boolean functions in the selection equation to create the dummy codes, but these could also be generated using the TRANSFORM function. See the TRANSFORM section in Chapter 2 for a description of Boolean operators.

Clicking the links below downloads the Blimp scripts and data for this example, and the full set of User Guide examples is available from a pull-down menu in the graphical interface.

[Ex7.9.imp](#) [data17.dat](#)

The syntax highlights are as follows.

- ❖ **ORDINAL** command identifies the binary dropout indicator
- ❖ **FIXED** command identifies a complete time score predictor

- ❖ LATENT command defines two between-cluster latent variables representing the random intercepts and slopes
- ❖ MODEL command uses labels ending in a colon to group models and order their summary tables on the output
- ❖ MODEL command estimates the random intercept and slope means using the `->` operator
- ❖ MODEL command sets the intercept of the regression equation equal to the level-2 latent mean (`1@beta0_j`)
- ❖ MODEL command omits the random coefficient listed after the vertical pipe
- ❖ MODEL command sets the random predictor's slope equal to the random coefficient (`time_i@beta1_j`)
- ❖ MODEL command specifies correlation between random intercepts and random slopes (level-2 latent variables)
- ❖ MODEL command includes Boolean functions that create time-specific dummy codes (e.g., `time_i==1`)
- ❖ MODEL command features product terms between the Boolean operators (dummy codes) and the level-2 latent variables

```

DATA: data17.dat;
VARIABLES: level2id time_i y_i dropout_i;
ORDINAL: dropout_i;
CLUSTERID: level2id;
MISSING: 999;
LATENT: level2id = beta0_j beta1_j;
FIXED: time_i;
MODEL:
latent.variables:
1 -> beta0_j beta1_j;
beta0_j ~~ beta1_j;
growth.model:
y_i ~ 1@beta0_j time_i@beta1_j;
missingness.model:
dropout_i ~ 1@0 (time_i == 0)@-3 (time_i == 1) (time_i == 2)
(time_i == 3) (time_i == 4) (time_i == 5)
(time_i > 0)*beta0_j (time_i > 0)*beta1_j | 1@0;
SEED: 90291;
BURN: 10000;
ITER: 10000;

```

The corresponding `rblimp` script is as follows.

```
library(rblimp)
load(file = 'data17.rda')

mymodel <- rblimp(
  data = data17,
  ordinal = 'dropout_i',
  clusterid = 'level2id',
  latent = 'level2id = beta0_j beta1_j',
  fixed = 'time_i',
  model = '
beta0_j ~ beta1_j;
growth.model:
y_i ~ 1@beta0_j time_i@beta1_j;
missingness.model:
dropout_i ~ 1@0 (time_i == 0)@-3 (time_i == 1) (time_i == 2)
(time_i == 3) (time_i == 4) (time_i == 5)
(time_i > 0)*beta0_j (time_i > 0)*beta1_j | 1@0',
  seed = 90291,
  burn = 10000,
  iter = 10000
)
output(mymodel)
```

8 Monte Carlo Studies in Blimp

Blimp also includes a `SIMULATE` command for creating artificial data sets with Monte Carlo computer simulation. This command creates a data set based on user-specified population parameters, after which it fits the equations from the `MODEL` command to the simulated data. The simulated data set can be saved by specifying `dataset = filename` with the `SAVE` command. Parameter estimates from the fitted model can also be saved following the procedure described in the `SAVE` section of Chapter 2. Note that each Blimp script generates a single artificial data set. To embed this functionality in a broader Monte Carlo computer simulation (e.g., inside a loop function), see Section 1.7. This chapter is currently under construction, but the `Examples` pull-down menu in Blimp Studio shows the following examples.

- ❖ 8.1: Simulation With Linear Regression
- ❖ 8.2: Simulation With Bivariate Regression
- ❖ 8.3: Simulation With Bifactor Model
- ❖ 8.4: Simulation With Random Intercept Model
- ❖ 8.5: Simulation With Random Slopes
- ❖ 8.6: Simulation With Linear Growth Model
- ❖ 8.7: Simulation with Two-level Decomposed Effects Model
- ❖ 8.8: Simulation With Missing Data Generation

9 References

- Alacam, E., Du, H., Enders, C. K., & Keller, B. T. (2023). A model-based approach to treating composite scores with missing items. *Psychological Methods*, Online first publication. <https://doi.org/10.1037/met0000584>.
- Albert, J. H., & Chib, S. (1993). Bayesian-Analysis of Binary and Polychotomous Response Data. *Journal of the American Statistical Association*, *88*(422), 669-679.
- Arnold, B. C., Castillo, E., & Sarabia, J. M. (2001). Conditionally specified distributions: An introduction. *Statistical Science*, *16*, 249-274.
- Asparouhov, T. (2006). General multi-level modeling with sampling weights. *Communications in Statistics—Theory and Methods*, *3*, 439-460.
- Asparouhov, T., & Muthén, B. (2021). Expanding the Bayesian structural equation, multilevel and mixture Models to logit, negative-binomial and nominal variables. (Advanced online publication), 1-16.
- Asparouhov, T., & Muthén, B. (2021). Advances in Bayesian model fit evaluation for structural equation models. *Structural Equation Modeling*, *28*, 1-14.
- Barnard, J., McCulloch, R., & Meng, X.-L. (2000). Modeling covariance matrices in terms of standard deviations and correlations, with application to shrinkage. *Statistica Sinica*, *10*, 1281-1311.
- Bartlett, J. W., Seaman, S. R., White, I. R., & Carpenter, J. R. (2015). Multiple imputation of covariates by fully conditional specification: Accommodating the substantive model. *Statistical Methods in Medical Research*, *24*(4), 462-487. doi:10.1177/0962280214521348
- Bates, D., Maechler, M., Bolker, B., Walker, S., Christensen, R. H. B., Singmann, H., . . . Krivitsky, P. N. (2021). Package 'lme4'. Retrieved from <https://cran.r-project.org/web/packages/lme4/>
- Bauer, D.J. (2017). A more general Model for testing measurement invariance and differential item functioning. *Psychological Methods*, *22*, 507-526

- Bodner, T. E. (2008). What improves with increased missing data imputations? *Structural Equation Modeling: A Multidisciplinary Journal*, *15*, 651-675.
- Carpenter, J. R., & Kenward, M. G. (2013). *Multiple imputation and its application*. West Sussex, UK: Wiley.
- Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Hillsdale, NJ: Erlbaum.
- Coxe, S., West, S.G., & Aiken, L.S. (2009). The analysis of count data: A gentle introduction to Poisson regression and its alternatives. *Journal of Personality Assessment*, *91*, 121-136.
- Diggle, P.D., & Kenward, M.G. (1994). Informative drop-out in longitudinal data analysis. *Journal of the Royal Statistical Society: Series C. Applied Statistics*, *43*, 49-93.
- Eekhout, I., Enders, C. K., Twisk, J. W. R., de Boer, M. R., de Vet, H. C. W., & Heymans, M. W. (2015). Analyzing Incomplete Item Scores in Longitudinal Data by Including Item Score Information as Auxiliary Variables. *Structural Equation Modeling-a Multidisciplinary Journal*, *22*(4), 588-602. doi:10.1080/10705511.2014.937670
- Enders, C. K. (2022). *Applied Missing Data* (2nd ed.). New York: Guilford Press.
- Enders, C. K. (2010). Missing not at random models for latent growth curve analyses. *Psychological Methods*, *16*, 1-16.
- Enders, C. K., Du, H., & Keller, B. T. (2020). A model-based imputation procedure for multilevel regression models with random coefficients, interaction effects, and other nonlinear terms. *Psychological Methods*, *25*, 88-112. doi:10.1037/met0000228
- Enders, C. K., & Gottschall, A. C. (2011). Multiple Imputation Strategies for Multiple Group Structural Equation Models. *Structural Equation Modeling-a Multidisciplinary Journal*, *18*(1), 35-54.
- Enders, C. K., & Keller, B. T. (2019). *Blimp Technical Appendix: Centering Covariates in a Bayesian Multilevel Analysis*. Retrieved from www.appliedmissingdata.com/multilevel-imputation.html:

- Enders, C. K., Keller, B. T., & Levy, R. (2018). A fully conditional specification approach to multilevel imputation of categorical and continuous variables. *Psychological Methods, 23*(2), 298-317. doi:10.1037/met0000148
- Enders, C. K., & Tofighi, D. (2007). Centering predictor variables in cross-sectional multilevel models: A new look at an old issue. *Psychological Methods, 12*, 121-138. doi:10.1037/1082-989X.12.2.121
- Enders, C. K., Vera, J. D., Keller, B. T., Lenartowicz, A., & Loo, S. K. (2024). *Building a simpler moderated nonlinear factor analysis model with Bayesian estimation*. Manuscript submitted for publication.
- Erler, N. S., Rizopoulos, D., Jaddoe, V. W., Franco, O. H., & Lesaffre, E. M. (2019). Bayesian imputation of time-varying covariates in linear mixed models. *Statistical Methods in Medical Research, 28*, 555-568. doi:10.1177/0962280217730851
- Erler, N. S., Rizopoulos, D., Rosmalen, J., Jaddoe, V. W., Franco, O. H., & Lesaffre, E. M. (2016). Dealing with missing covariates in epidemiologic studies: a comparison between multiple imputation and a full Bayesian approach. *Statistics in Medicine, 35*(17), 2955-2974. doi:10.1002/sim.6944
- Geldhof, G. J., Anthony, K. P., Selig, J. P., & Mendez-Luck, C. A. (2018). Accommodating binary and count variables in mediation: A case for conditional indirect effects. *International Journal of Behavioral Development, 42*(2), 300-308.
- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2014). *Bayesian data analysis* (3rd ed.). Boca Raton, FL: CRC Press.
- Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science, 7*, 457-472. doi:10.1214/ss/1177011136
- Goldstein, H. (2011). *Multilevel statistical models* (4th ed.) West Sussex, UK: Wiley.
- Goldstein, H., Carpenter, J., Kenward, M. G., & Levin, K. A. (2009). Multilevel models with multivariate mixed response types. *Statistical Modelling, 9*(3), 173-197. doi:10.1177/1471082x0800900301
- Gomer, K., & Yuan, K.-H. (2021). Subtypes of the missing not at random missing data mechanism. *Psychological Methods*, Advanced online publication, 1-40.
- Graham, J. W. (2009). Missing data analysis: making it work in the real world. *Annual Review of Psychology, 60*, 549-576. doi:10.1146/annurev.psych.58.110405.085530

- Graham, J. W., Olchowski, A. E., & Gilreath, T. D. (2007). How many imputations are really needed? Some practical clarifications of multiple imputation theory. *Prevention Science, 8*(3), 206-213. doi:10.1007/s11121-007-0070-9
- Grund, S., Lüdke, O., & Robitzsch, A. (2016). Multiple imputation of missing covariate values in multilevel models with random slopes: a cautionary note. *Behavior Research Methods, 48*(2), 640-649. doi:10.3758/s13428-015-0590-3
- Grund, S., Robitzsch, A., & Lüdke, O. (2021). Package 'mitml'. Retrieved from cran.r-project.org/web/packages/mitml/
- Hamaker, E. L., Kuiper, R. M., & Grasman, R. P. (2015). A critique of the cross-lagged panel model. *Psychological Methods, 20*(1), 102-116.
- Hamaker, E. L., & Muthén, B. (2019). The fixed versus random effects debate and how it relates to centering in multilevel modeling. *Psychological Methods*.
- Harel, O. (2007). Inferences on missing information under multiple imputation and two-stage multiple imputation. *Statistical Methodology, 4*(1), 75-89. doi:10.1016/j.stamet.2006.03.002
- Hedeker, D. & Gibbons, R. (1997). Application of random-effects pattern-mixture models for missing data in longitudinal studies. *Psychological Methods, 2*, 64-78.
- Hedeker, D. & Mermelstein, R.J., & Demirtas, H. (2008). An application of a mixed-effects location scale model for analysis of ecological momentary assessment (EMA) data. *Biometrics, 64*, 627-634.
- Ibrahim, J. G., Chen, M. H., & Lipsitz, S. R. (2002). Bayesian methods for generalized linear models with covariates missing at random. *Canadian Journal of Statistics-Revue Canadienne De Statistique, 30*(1), 55-78. doi:10.2307/3315865
- Ibrahim, J. G., Lipsitz, S. R., & Chen, M. H. (1999). Missing covariates in generalized linear models when the missing data mechanism is non-ignorable. *Journal of the Royal Statistical Society: Series B (Statistical Methodology), 61*(1), 173-190. doi:10.1111/1467-9868.00170
- Johnson, P.E.. (2019). Package 'rockchalk'. Retrieved from <https://cran.r-project.org/web/packages/rockchalk/rockchalk.pdf>
- Johnson, V. E., & Albert, J. H. (1999). *Ordinal data modeling*. New York: Springer.

- Jorgensen, T. D. (2024). *lavaan.mi*. Fit Structural Equation Models to Multiply Imputed Data [Computer software]. R package version 0.1-0.0029, retrieved from <https://github.com/TDJorgensen/lavaan.mi>
- Kasim, R. M., & Raudenbush, S. W. (1998). Application of Gibbs sampling to nested variance components models with heterogeneous within-group variance. *Journal of Educational and Behavioral Statistics*, 32, 93-116.
- Keller, B. T. (2022). An introduction to factored regression models with Blimp. *Psych*, 4, 10-37.
- Keller, B. T. (2024). A general approach to modeling latent variable interactions and nonlinear effects. Manuscript submitted for publication.
- Keller, B. T., & Enders, C. K. (2021). An investigation of factored regression missing data methods for multilevel models with cross-level interactions. *Manuscript submitted for publication*.
- Kim, S., Belin, T. R., & Sugar, C. A. (2018). Multiple imputation with non-additively related variables: Joint-modeling and approximations. *Statistical Methods in Medical Research*, 27(6), 1683-1694. doi:10.1177/0962280216667763
- Kim, S., Sugar, C. A., & Belin, T. R. (2015). Evaluating model-based imputation methods for missing covariates in regression models with interactions. *Statistics in Medicine*, 34(11), 1876-1888. doi:10.1002/sim.6435
- Levy, R., & Enders, C. (2021). Full conditional distributions for Bayesian multilevel models with additive or interactive effects and missing data on covariates. *Communications in Statistics—Simulation and Computation, Advanced online publication*, 1-25.
- Levy, R., & McNeish, D. (2023). Perspectives on Bayesian inference and their implications for data analysis. *Psychological Methods*, 28(3), 719-739.
- Lipsitz, S. R., & Ibrahim, J. G. (1996). A conditional model for incomplete covariates in parametric regression models. *Biometrika*, 83(4), 916-922. doi:DOI 10.1093/biomet/83.4.916
- Little, R. (2009). Selection and pattern-mixture models. In G. Fitzmaurice, M. Davidian, G. Vebeke, & G. Molenberghs (Eds.), *Longitudinal Data Analysis* (pp. 409-431). Boca Raton: Chapman & Hall.

- Liu, J. C., Gelman, A., Hill, J., Su, Y. S., & Kropko, J. (2014). On the stationary distribution of iterative imputations. *Biometrika*, *101*(1), 155-173.
- Liu, H. Y., Zhang, Z. Y., & Grimm, K. J. (2016). Comparison of inverse Wishart and separation-strategy priors for Bayesian estimation of covariance parameter matrix in growth curve analysis. *Structural Equation Modeling: A Multidisciplinary Journal*, *23*, 354–367.
- Longford, N. (1989). Contextual effects and group means. *Multilevel Modelling Newsletter*, *1*(3), 5.
- Lüdtke, O., Marsh, H. W., Robitzsch, A., Trautwein, U., Asparouhov, T., & Muthén, B. (2008). The Multilevel Latent Covariate Model: A New, More Reliable Approach to Group-Level Effects in Contextual Studies. *Psychological Methods*, *13*(3), 201-229. doi:10.1037/a0012869
- Lüdtke, O., Robitzsch, A., & West, S. G. (2020a). Analysis of interactions and nonlinear effects with missing data: a factored regression modeling approach using maximum likelihood estimation. *Multivariate Behavioral Research*, *55*(3), 361-381.
- Lüdtke, O., Robitzsch, A., & West, S. G. (2020b). Regression models involving nonlinear effects with missing data: A sequential modeling approach using Bayesian estimation. *Psychological Methods*, *25*, 157-181.
- Mackinnon, D. P. (2008). *Introduction to statistical mediation analysis*. New York: Lawrence Erlbaum Associates.
- McNeish, D. (2021). Specifying location-scale models for heterogeneous variances as multilevel SEMs. *Organizational Research Methods*, *24*(3), 630–653.
- McNeish, D., & Hamaker, E. L. (2020). A primer on two-level dynamic structural equation models for intensive longitudinal data in *Mplus*. *Psychological Methods*, *25*(5), 610–635.
- Merkle, E. C., & Rosseel, Y. (2018). blavaan: Bayesian structural equation models via parameter expansion. *Journal of Statistical software*, *85*(4), 1-30. doi:10.18637/jss.v085.i04.

- Mulder, J. D., & Hamaker, E. L. (2021). Three extensions of the random intercept cross-lagged panel model. *Structural Equation Modeling: A Multidisciplinary Journal*, 28(4), 638–648.
- Muthén, B., Asparouhov, T., Hunter, A.M., & Leuchter, A.F. (2011). Growth modeling with nonignorable dropout: Alternative analyses of the STAR*D antidepressant trial. *Psychological Methods*, 16, 17–33.
- Muthén, B., Muthén, L., & Asparouhov, T. (2016). *Regression and mediation analysis using Mplus*. Los Angeles, CA.: Muthén & Muthén.
- Olsen, M. K., & Schafer, J. L. (2001). A two-part random-effects model for semicontinuous longitudinal data. *Journal of the American Statistical Association*, 96(454), 730–745.
- Polson, N. G., Scott, J. G., & Windle, J. (2013). Bayesian inference for logistic models using Pólya–Gamma latent variables. *Journal of the American Statistical Association*, 108(504), 1339–1349.
- Raudenbush, S. W., & Bryk, A. S. (2002). *Hierarchical linear models: Applications and data analysis methods* (2nd ed.). Thousand Oaks, CA: Sage.
- Rights, J. D., & Sterba, S. K. (2019). Quantifying explained variance in multilevel models: An integrative framework for defining R-squared measures. *Psychological Methods*, 24, 309–338.
- Rosseel, Y., Jorgensen, T. D., & Rockwood, N. J. (2021). *Package 'lavaan'*. <https://CRAN.R-project.org/package=lavaan>.
- Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys*. New York: Wiley.
- Seaman, S. R., Bartlett, J. W., & White, I. R. (2012). Multiple imputation of missing covariates with nonlinear effects and interactions: an evaluation of statistical methods. *BMC Medical Research Methodology*, 12, 46. doi:10.1186/1471-2288-12-46
- van Buuren, S. (2007). Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research*, 16(3), 219–242. doi:10.1177/0962280206074463
- van Buuren, S., Brand, J. P. L., Groothuis-Oudshoorn, C. G. M., & Rubin, D. B. (2006). Fully conditional specification in multivariate imputation. *Journal of Statistical*

- Computation and Simulation*, 76(12), 1049-1064.
doi:10.1080/10629360600810434
- van Buuren, S., & Groothuis-Oudshoorn, K. (2011). MICE: Multivariate imputation by chained equations in R. *Journal of Statistical software*, 45, 1-67.
doi:10.18637/jss.v045.i03
- von Hippel, P. T. (2018). How Many Imputations Do You Need? A Two-stage Calculation Using a Quadratic Rule. *Sociological Methods & Research*, 0049124117747303.
- Wu, M.C., & Carroll, R.J. (2018). Estimation and comparison of changes in the presence of informative right censoring by modeling the censoring process. *Biometrics*, 44, 175-188.
- Yaremych, H. E., & Preacher, K. J. (2021). Centering categorical predictors in multilevel models: Best practices and interpretation.
- Yeo, I. K., & Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4), 954-959.
- Yuan, Y., & MacKinnon, D. P. (2009). Bayesian mediation analysis. *Psychological Methods*, 14(4), 301-322.
- Zhang, Q., & Wang, L. (2017). Moderation analysis with missing data in the predictors. *Psychological Methods*, 22(4), 649-666. doi:10.1037/met000010