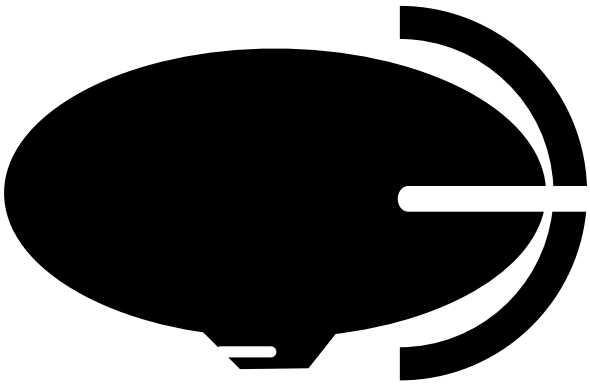
Blimp User's Guide

Version 1.1



Brian T. Keller
bkeller2@ucla.edu
Craig K. Enders
cenders@psych.ucla.edu

March 2018

Developed by Craig K. Enders and Brian T. Keller.

Blimp was developed with funding from Institute of Educational Sciences award R305D150056.

Craig K. Enders, P.I. Email: cenders@psych.ucla.edu.

Programming by Brian T. Keller. Email: bkeller2@ucla.edu

GUI programming by Nitish Mehta and Brian T. Keller

For support please email blimp.imputation@gmail.com.

Any published work derived from the use of this software, please cite the following sources:

Enders, C. K., Keller, B. T., & Levy, R. (2017). A chained equations imputation approach for multilevel data with categorical and continuous variables. Psychological Methods, Advance online publication. http://dx.doi.org/10.1037/met0000148.

Keller, B. T., & Enders, C. K. (2018). Blimp User's Manual (Version 1.1). Los Angeles, CA.

Contents

1 Introduction **1**

2 Quick Start Guide **2**

Example 2.1: FCS Imputation 2

Example 2.2: SMC-FCS Imputation 5

Running Blimp 6

Blimp Output 7

3 Blimp Command Language **9**

General Conventions 9

Reading Data 9

The Outcome and Model Commands 10

The `OUTCOME` Command 10

The `MODEL` Command 11

Specifying Random Slopes 11

Specifying Interaction and Polynomial Terms 12

Algorithmic Options 13

`OPTIONS` Keywords 14

4 Running Blimp **17**

5 Blimp Output **18**

Multiple Chains 20

6 Convergence Diagnostics with Blimp **22**

7 Analysis Examples **24**

Example 7.1: Single-Level Regression Analysis 25

Diagnosing Convergence for Example 7.1 25

Separate Format for R, SAS, SPSS, and Stata Analysis 27

Example 7.2: Single-Level Regression with Interaction Effect 27

Example 7.3: Two-Level Regression with Random Intercepts 29

Example 7.4: Two-Level Regression with Random Slopes 29
Example 7.5: Two-Level Regression with Random Slopes and a Cross-Level Interaction Effect . . . 31
Example 7.6: Three-Level Regression with Random Intercepts 31
Example 7.7: Three-Level Regression with Random Slopes and Cross-Level Interaction Effect . . . 32

8 Running a Simulation **34**
Internal Blimp Simulations 34
Output from `SIMULATE` command 36
External Blimp Simulations 37

9 Error and Warning Message Reference **39**
Error Messages 39
Warning Messages 45

References **49**

1 Introduction

Blimp is a software program designed to perform multiple imputation via fully conditional specification (FCS) and substantive model-compatible FCS (SMC-FCS; Bartlett, Seaman, White, & Carpenter, 2014). FCS is appropriate for many general situations, whereas SMC-FCS is useful for single- and multilevel models with incomplete interaction effects or polynomial terms and multilevel models with random slopes. Blimp can perform single-level and multilevel imputation (up to three levels) with mixtures of continuous (normal), ordinal, and nominal variables, although substantive model-compatible imputation is currently limited to continuous and ordinal (including binary) variables. Blimp can accommodate a range of common analysis models, including models with (a) random intercepts and random slopes, (b) different within- and between-cluster covariance structures (e.g., contextual effects analyses, multilevel structural equation models), and (c) heterogeneous level-1 variance structures (e.g., daily diary data). Technical details of the Blimp FCS algorithm are available in:

Enders, C. K., Keller, B. T., & Levy, R. (2017). A chained equations imputation approach for multilevel data with categorical and continuous variables. Psychological Methods, Advance online publication. http://dx.doi.org/10.1037/met0000148.

The manuscript is available upon request or from www.appliedmissingdata.com. Blimp scripts and analysis examples for M*plus*, R, SAS, SPSS, and Stata are also available from the same URL.

2 Quick Start Guide

This chapter provides a brief tutorial to help users quickly begin using the Blimp command language. Documents illustrating the graphical user interface (GUI) are available at www.appliedmissingdata.com. Example 2.1 and Example 2.2 below depict prototypical Blimp scripts for two-level imputation (single-level imputation is nearly identical). We use these examples throughout the remainder of the chapter, and all information in the Quick Start Guide is presented in more detail in subsequent chapters.

Example 2.1: FCS Imputation

Example 2.1: Quick Start Example for Fully Conditional Specification

```
# This is a comment

DATA: /Users/name/Desktop/mydata.csv;

VARIABLES: id av y x1 x2 x3

           w1 w2 w3;

ORDINAL: x2 w2;

NOMINAL: x3 w3;

MODEL: id ~ av y:x1 y:x2 x3 w1 w2 w3;

BURN: 2000;

THIN: 500;

NIMPS: 10;

MISSING: -99;

SEED: 90291;

OUTFILE: /Users/name/Desktop/myimps*.csv;

OPTIONS: separate;
```

Blimp commands can be entered in the syntax editor of the GUI or in a plain text file (raw ASCII format) with a '.imp' extension. The Blimp syntax uses the following conventions: (a) the program is not case sensitive, (b) command names (e.g., `DATA`, `VARIABLES`, etc.) are followed by a colon (`:`), (c) each command is terminated with a semicolon (`;`), (d) commands can span multiple lines (e.g., the `VARIABLES` command in Example 2.1), (e) a dash can be used to specify a range of variables with the same prefix and/or suffix across a range of numbers (e.g., `x1-x3` is the same as `x1 x2 x3`, `x1a-x3a` is the same as `x1a x2a x3a`, etc.), and (f) the number sign (`#`) is used to create a comment line that Blimp ignores.

The `DATA`, `VARIABLES`, `ORDINAL`, `NOMINAL`, and `MISSING` commands specify features of the raw data file. The `DATA` command specifies the full file path to the input data set. When the file path includes spaces, the file path should not be enclosed quotations (the file path in Example 2.1 is consistent with macOS). The input data set must be saved as a CSV (comma separated values) format or a whitespace delimited file (including tab delimited files). Note that some statistical software packages (e.g., SPSS) that save text files in Unicode format embed hidden characters that cause the import to fail. This can typically be changed in the program's preferences (e.g., in SPSS, set the output text format to Locale). The `VARIABLES` command specifies the variables in the input file. The variable names must be alphanumeric with no periods or other special punctuation marks. A dash can be used to specify a range of variables with the same prefix and/or suffix across a range of numbers (e.g., `x1-x3` is the same as `x1 x2 x3`, `x1a-x3a` is the same as `x1a x2a x3a`, etc.), but dashes are limited to this case. All missing values must be assigned the same numeric code, the value of which is specified via the `MISSING` command.

The `ORDINAL` and `NOMINAL` commands designate categorical variables (unless otherwise specified, Blimp assumes a normal distribution for all variables). To maximize computational speed, we recommend specifying binary variables on the `ORDINAL` line, although the underlying statistical model invoked by the `NOMINAL` command is equivalent in this case. Nominal variables must be represented as a single variable with numeric codes. Complete variables listed on the `NOMINAL` line are automatically recoded into a set of dummy codes during imputation. Example 2.1 depicts an imputation model with two ordinal variables (`x2` and `w2`) and two nominal variables (`x3` and `w3`).

The `MODEL` command specifies the variables in the imputation regression models. The FCS algorithm cycles through incomplete variables one at a time, imputing each variable from a regression equation that conditions on all other variables listed on the `MODEL` line. At a minimum, the `MODEL` statement should include all variables and effects of interest in the analysis model(s), but the list may also include additional

auxiliary variables. Note that the variable list should include both complete and incomplete variables, and the user need not specify which variables have missing values. Additionally, Blimp will automatically determine the level at which a variable is measured in a multilevel data set. For single-level imputation, the imputation model variables are listed to the right of the tilde symbol. Multilevel imputation requires a cluster-level identifier variables to the left of the tilde symbol. Two-level imputation requires a single identifier for the level-2 sampling unit (cluster), and three-level imputation requires level-2 and level-3 identifier variables. The order of the identifier variables does not matter. Example 2.1 depicts a two-level imputation problem, where `id` is the level-2 identifier variable.

Blimp can accommodate multilevel analyses with random slopes. To specify a random slope between two variables, simply place a colon (`:`) between two variables instead of a space (see Chapter 3). In Example 2.1, the imputation model includes random slopes for `y` and `x1` and `y` and `x2`. Although the standard FCS algorithm makes no reference to a variable's role in the subsequent analysis model, it would be unusual and unnecessary to specify random relations among predictor variables. Thus, any random slope specification should always include an outcome variable from the analysis model. For example, the previous syntax corresponds to an analysis where `y` is the outcome and `x1` and `x2` are level-1 predictors with random coefficients. Although both FCS and SMC-FCS can accommodate random slopes, we recommend the latter (described below) because it maximizes accuracy of the between-cluster variance components.

The `BURN`, `THIN`, `SEED`, `NIMPS`, and `OPTIONS` commands specify features of the MCMC algorithm. In Example 2.1 the `BURN` command is set to 2000, meaning that 2000 iterations are performed prior to saving the first imputed data set. The `THIN` command specifies the so-called between-imputation interval. In Example 2.1, imputed data sets are saved after every 500^{th} iteration. The `NIMPS` command specifies the number of imputations desired. The `SEED` command specifies the seed value for Blimp's pseudorandom number generator. The `OPTIONS` command specifies various miscellaneous options, such as the format of the imputed data sets (e.g., imputations stacked in a single file or output to separate files), the prior distributions used for variance components, etc. Section presents more information about the specific options and their defaults.

Finally, the `OUTFILE` command specifies the complete file path for the resulting imputed data set(s). Example 2.1 saves each imputed data set to a separate file. This specification requires an asterisk in the file name, and this asterisk is subsequently replaced by a numeric value (e.g., myimps1.dat, myimps2.dat, etc.).

Example 2.2: SMC-FCS Imputation

As noted previously, the standard FCS algorithm cycles through incomplete variables one at a time, making no reference to a variable's role in the subsequent analysis. That is, the imputation process applies the same procedure to explanatory variables and outcomes. In contrast, SMC-FCS treats predictor variables and outcomes somewhat differently: explanatory variables are imputed conditional on other predictors and auxiliary variables (but not the outcome), and the outcome is imputed conditional on the predictors and possible auxiliary variables. Although explanatory variables are imputed from a model that does not include the outcome, a Metropolis sampling step selects imputations that are consistent with the substantive analysis model. Example 2.2 below illustrates a prototypical SMC-FCS setup.

Example 2.2: Quick Start Example for Substantive Model-Compatible Imputation

```
# This is a comment

DATA: /Users/name/Desktop/mydata.csv;

VARIABLES: id av y x1 x2 x3

    w1 w2 w3;

ORDINAL: x2 x3 w2 w3;

MODEL: id ~ av y:x1 y:x2 x3 w1 w2 w3 x1*w3;

OUTCOME: y;

BURN: 2000;

THIN: 500;

NIMPS: 10;

MISSING: -99;

SEED: 90291;

OUTFILE: /Users/name/Desktop/myimps*.csv;

OPTIONS: separate;
```

The SMC-FCS syntax is nearly identical to standard FCS, with two important exceptions. First, SMC-FCS requires the `OUTCOME` command, which is used to specify the dependent variable from the substantive analysis model. When specifying SMC-FCS, all other variables on the `MODEL` line are assumed to function

as explanatory variables in the analysis model or as auxiliary variables. Second, SMC-FCS allows the user to model interactive or polynomial terms. To illustrate, the syntax in Example 2.1 includes an interaction between `x1` and `w3`, which is specified by joining these two variables with an asterisk (i.e., `x1*w2`). Importantly, the interaction is not imputed directly, nor is it computed passively (e.g., by imputing lower-order terms and subsequently forming the product from the imputed values). Rather, a special Metropolis sampling step selects imputations that are consistent with an analysis model that includes the product. Descriptions of this procedure are found in Bartlett, Seaman, White, and Carpenter (2015) and Zhang and Wang (2016), and Chapter 3 gives additional details on interactive effects.

SMC-FCS is currently available with continuous and ordinal (or binary) variables, and support for nominal variables will be added in a later release. Whenever possible, we recommend SMC-FCS imputation for analyses that include interactive or polynomial terms and/or random slopes, as simulation and analytic work suggests that SMC-FCS is superior to FCS in these cases. With the exception of these two scenarios, we would often expect similar performance from FCS and SMC-FCS.

Finally, note that product or polynomial terms specified on the `MODEL` command of SMC-FCS do not appear in the output data files. These variables must be computed prior to analysis. For examples, see the analysis scripts available at www.appliedmissingdata.com.

Running Blimp

Blimp is executed via the 'Run' pull-down option from the GUI or via the command-line. A terminal emulator (e.g., Terminal.app in macOS, cmd.exe in Windows) is used to run Blimp from the command-line. To run Blimp via a command-line, type blimp followed by the full file path to the syntax file. For macOS, the command-line tools must be installed first. See Chapter 4 for more information. The command-line specification for macOS is as follows:

```
$ blimp ~/desktop/InputExample.imp
```

where 'InputExample.imp' is a Blimp syntax script saved in the user's Desktop directory. Most terminal applications automatically generate the appropriate file paths if you drag the application and syntax files into the terminal window.

Blimp Output

After executing Blimp (GUI or shell), a header is printed to the output window. This header provides the version number along with contact information. An example header is shown below.

```
                                Blimp
                                 v1.0

          Developed by Craig K. Enders and Brian T. Keller.
                Blimp was developed with funding from
        Institute of Educational Sciences award R305D150056.
        Craig K. Enders, P.I. Email: cenders@psych.ucla.edu.
      Programming by Brian T. Keller. Email: bkeller2@ucla.edu
                There is no expressed license given.


             DO NOT DISTRIBUTE WITHOUT WRITTEN CONSENT
```

Below the header, Blimp lists the algorithmic options specified in the syntax file. In addition to specified options, the list includes default settings that may not have been explicitly specified in the syntax. For example, in Example 2.1 the keywords `hov` and `clmean` were specified in the `OPTIONS` command, so these options are listed in the output. The syntax did not specify prior distributions for variance components, thus triggering appropriate default settings.

```
ALGORITHMIC OPTIONS SPECIFIED:

  Imputation method:                Fully conditional specification
  MCMC algorithm:                   MICE sampler (MICE)
  Between-cluster imputation model: Cluster means included (CLMEAN)
  Residual variance structure:      Homogeneous level-1 variance (HOV)
  Prior for covariance matrices:    Identity matrix, df = p + 1 (PRIOR1)
  Prior for residual variance:      Unit sum of squares, df = 2 (PRIOR1)
  Diagnostics:                      No potential scale reduction (NOPSR)
  Imputation format:                Separate files (SEPARATE)
```

Following the algorithmic options, Blimp displays a summary of the imputation model variables.

```
VARIABLES IN IMPUTATION MODEL:

  Level-2 identifier:     id
  Complete variables:     av x3#1 x3#2 x3#3
  Incomplete continuous:  y x1 w1
  Incomplete ordinal:     x2 w2
  Incomplete nominal:     w3
```

Notice that x3 is listed three times as a complete variable (i.e., x3#1 x3#2 x3#3). Because x3 is a 4-category nominal variable, Blimp automatically recodes the variable into three dummy codes, where the first group (the lowest numeric code) serves as the reference.

Next, the output displays the progress of the MCMC algorithm. Example 2.1 requests 2000 burn-in iterations, and the program issues a message when it has completed 25%, 50%, 75%, and 100% of these initial iterations. Following the burn-in phase, Blimp prints a timestamp as it saves imputed data sets.

```
ITERATION HISTORY:

Starting Burn-in on Fri Sep  8 10:07:35 2017
    Burn-in iteration 500 complete on Fri Sep  8 10:07:37 2017
    Burn-in iteration 1000 complete on Fri Sep  8 10:07:40 2017
    Burn-in iteration 1500 complete on Fri Sep  8 10:07:43 2017
    Burn-in iteration 2000 complete on Fri Sep  8 10:07:46 2017
Burn-in complete on Fri Sep  8 10:07:46 2017
    Imputation Saved 1 on Fri Sep  8 10:07:46 2017
    Imputation Saved 2 on Fri Sep  8 10:07:49 2017
    Imputation Saved 3 on Fri Sep  8 10:07:53 2017
    Imputation Saved 4 on Fri Sep  8 10:07:56 2017
    Imputation Saved 5 on Fri Sep  8 10:08:00 2017
    Imputation Saved 6 on Fri Sep  8 10:08:03 2017
    Imputation Saved 7 on Fri Sep  8 10:08:08 2017
    Imputation Saved 8 on Fri Sep  8 10:08:12 2017
    Imputation Saved 9 on Fri Sep  8 10:08:16 2017
    Imputation Saved 10 on Fri Sep  8 10:08:20 2017
```

Finally, Blimp prints the order of the variables in the imputed data set(s). All variables listed on the VARIABLES command are saved to the file(s), regardless of whether the variables are used in imputation.

```
VARIABLE ORDER IN SAVED DATA:

   id av y x1 x2 x3 w1 w2 w3
```

Variables are saved in the same order as they appear on the VARIABLES command, with one important exception. When imputations are stacked in a single file (the stacked keyword of the OPTION command), the first variable in the file is an identifier variable that indexes the imputed data sets. In this case, the variable order appears as follows.

```
VARIABLE ORDER IN SAVED DATA:

  imp# id av y x1 x2 x3 w1 w2 w3
```

3 Blimp Command Language

This Chapter gives a more detailed account of the Blimp command language illustrated previously in the Quick Start Chapter. Blimp commands can be entered in the syntax editor of the GUI, or in a plain text file (raw ASCII format) with a '.imp' extension. The main commands used in a standard Blimp input and are listed below.

General Conventions

The Blimp syntax uses the following conventions: (a) the program is not case sensitive, (b) command names (e.g., `DATA`, `VARIABLES`, etc.) are followed by a colon (`:`), (c) each command is terminated with a semicolon (`;`), (d) commands can span multiple lines (e.g., the `VARIABLES` command in Example 2.1), (e) a dash can be used to specify a range of variables with the same prefix and/or suffix across a range of numbers (e.g., `x1-x3` is the same as `x1 x2 x3`, `x1a-x3a` is the same as `x1a x2a x3a`, etc.), and (f) the number sign (`#`) is used to create a comment line that Blimp ignores.

Reading Data

`DATA`. The `DATA` command specifies the full file path to the input data set. When the file path includes spaces, the path should not be enclosed quotations. Blimp accepts a comma separated value (CSV) file or a whitespace delimited file (including tab delimited files). Data must be provided in plain text with no header. It is important to note that Blimp currently accepts only numerical values in the data sets. Note that statistical software packages that save text files in Unicode format (e.g., SPSS) embed hidden characters that cause the import to fail. This can typically be changed in the program's preferences (e.g., in SPSS, set the output text format to Locale).

`VARIABLES`. The `VARIABLES` command specifies the variables in the input file. The variable names must be alphanumeric with no periods or other special punctuation marks. The variable list may include variables that are not used in the imputation regression models. All variables listed on the `VARIABLES` line will appear in the imputed data sets returned by Blimp. A dash can be used to specify a range of

variables with the same prefix and/or suffix across a range of numbers (e.g., `x1-x3` is the same as `x1 x2 x3`, `x1a-x3a` is the same as `x1a x2a x3a`, etc.), but dashes are limited to this case.

ORDINAL. The `ORDINAL` command specifies incomplete ordinal (including binary) variables. For computational efficiency, we recommend listing binary variables on the `ORDINAL` line, but these variables could also be treated as nominal. A dash can be used to specify a range of variables with the same prefix and/or suffix across a range of numbers (e.g., `x1-x3` is the same as `x1 x2 x3`, `x1a-x3a` is the same as `x1a x2a x3a`, etc.), but dashes are limited to this case.

NOMINAL. The `NOMINAL` command specifies nominal variables, incomplete or complete. Nominal variables must be represented as a single variable with numeric codes. Complete nominal variables are automatically recoded into a set of dummy codes during imputation. A dash can be used to specify a range of variables with the same prefix and/or suffix across a range of numbers (e.g., `x1-x3` is the same as `x1 x2 x3`, `x1a-x3a` is the same as `x1a x2a x3a`, etc.), but dashes are limited to this case.

MISSING. The `MISSING` command is used to specify the numeric code that represents a missing value in the input data set. All missing values must be coded with a single numeric value (e.g., -99).

The Outcome and Model Commands

As noted previously, the standard FCS algorithm cycles through incomplete variables one at a time, making no reference to a variable's role in the subsequent analysis. That is, the imputation process applies the same procedure to explanatory variables and outcomes. As such, the user simply needs to invoke the `MODEL` command. In contrast, SMC-FCS treats predictor variables and outcomes somewhat differently: explanatory variables are imputed conditional on other predictors and auxiliary variables (but not the outcome), and the outcome is imputed conditional on the predictors and possibly auxiliary variables. SMC-FCS requires both the `OUTCOME` and the `MODEL` commands.

The **OUTCOME** Command

Specifying the `OUTCOME` command automatically invokes SMC-FCS imputation and its requisite algorithmic options (e.g., a Metropolis sampler). This command lists a single variable that functions as the dependent variable in a subsequent statistical analysis (single-level or multilevel), as follows.

```
OUTCOME: y;
```

As noted previously, SMC-FCS is currently limited to continuous (normal) and ordinal (including binary) variables. Functionality for nominal variables will be added in a later release.

The `MODEL` Command

The `MODEL` command specifies the variables that are included in the imputation regression models. At a minimum, this list must include all variables and effects of interest in the analysis model(s), but the list may also include additional auxiliary variables. Note that the variable list should include both complete and incomplete variables, and the user need not specify which variables have missing values. Additionally, Blimp will automatically determine the level at which a variable is measured in a multilevel data set. A dash can be used to specify a range of variables with the same prefix and/or suffix across a range of numbers (e.g., `x1-x3` is the same as `x1 x2 x3`, `x1a-x3a` is the same as `x1a x2a x3a`, etc.).

The `MODEL` statement for single-level imputation consists of a tilde symbol followed by a variable list, as follows.

~ Variables included in imputation

The `MODEL` statement for multilevel imputation contains two parts: an identifier part and a model part. A model statement takes the following form

Identifier variables ~ Variables included in imputation

Two-level imputation requires a cluster-level identifier variable before the tilde (e.g., in a data set with students nested in schools, the school identifier code), and three-level imputation requires level-2 and level-3 identifier variables. The order of the identifier variables does not matter. The following example illustrates the `MODEL` statement for a two-level imputation problem. The single identifier variable `idvar` to the left of the tilde indicates a two-level data structure, and the variables included in imputation are `x`, `y`, and `z`.

```
MODEL: idvar ~ x y z;
```

Specifying Random Slopes

Blimp automatically includes random intercepts for all incomplete variables in a multilevel data set. The FCS and SMC-FCS algorithms can also accommodate multilevel analyses with random slopes. To specify a random slope between two variables, place a colon (`:`) between the variables instead of a space.

Returning to the previous example, suppose that x and y have a random association in the analysis model. The imputation model for this problem would specify a random slope as follows.

```
MODEL: idvar ~ y:x z;
```

If the outcome variable y is to have a random slope with another variable, z, then y is repeated twice in the model statement, as follows:

```
MODEL: idvar ~ y:x y:z;
```

As noted elsewhere, we recommend SMC-FCS for imputing random slopes, as this approach maximizes accuracy of the variance components.

Specifying Interaction and Polynomial Terms

Traditional imputation schemes typically force the user to impute incomplete product or polynomial terms (e.g., the so-called "just another variable" approach). This strategy is not ideal because it imposes incorrect distributional assumptions on the product term and is prone to bias. SMC-FCS imputation offers an alternative approach that appears to be far superior. Rather than imputing the product directly, a special Metropolis sampling step selects imputations that are consistent with an analysis model that includes the specified product.

To illustrate the specification of a product term, consider a two-level analysis that includes a random slope for a level-1 predictor x and a cross-level interaction involving x and a level-2 predictor w. The Blimp syntax for SMC-FCS requires the OUTCOME command and the following specification on the MODEL line.

```
MODEL: idvar ~ y:x w x*w;
```

Notice that the variables forming the interaction are joined with an asterisk, i.e. x*w. This specification extends to accommodate higher-order interaction terms as well. For example, consider a single-level analysis with a three-way interaction and all corresponding lower-order terms. The MODEL statement would appear as follows.

```
MODEL: ~ y x z w x*z x*w z*w x*z*w;
```

Polynomial terms are specified using a similar convention. For example, an analysis that expresses the outcome variable as a quadratic function of a predictor would have the following MODEL command.

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

Importantly, interaction and polynomial terms are not imputed directly, nor are they computed passively (e.g., by imputing lower-order terms and subsequently forming the product from the imputed values). Descriptions of the SMC-FCS procedure are found in Bartlett et al. (2015) and Zhang and Wang (2016).

Interaction and polynomial terms implemented with SMC-FCS do not appear in the output data sets and must be computed prior to analysis. Examples of interactive effects along with M*plus*, SAS, SPSS, and Stata analysis scripts are available from www.appliedmissingdata.com.

Algorithmic Options

BYGROUP. The BYGROUP command is used to perform imputation separately for subgroups in the data. Among other things, this specification is useful when the analysis model posits interactions involving a categorical moderator variable that is completely observed, or for psychometric analyses that posit unique covariance matrices across groups (e.g., analyses investigating measurement invariance). For example, consider a scenario involving an intervention, where the intervention status is coded as a dummy variable. Listing intervention status on the BYGROUP command would preserve all possible two-way associations between intervention status and the variables listed on the MODEL command. Only a single categorical variable is allowed on the BYGROUP command, although higher-order associations can be preserved by coding multiple categorical variables into a single variable, sample size permitting. Importantly, the variable listed on the BYGROUP command should not be listed on the ORDINAL, NOMINAL, or MODEL lines. Also, multiple MCMC chains (see discussion of the CHAINS command) are not allowed when BYGROUP is used.

BURN. The BURN command is used to specify the number of burn-in iterations that are performed prior to saving the first imputed data set. The number of iterations should be determined by the convergence diagnostics addressed in Chapter 6.

THIN. The THIN command is used to specify the thinning (between-imputation) interval, which is the number of iterations separating each data set. For example, a value of 1000 would generate an imputed data set after every 1000th computational cycle. Again, the convergence diagnostics discussed in Chapter 6 can be useful for specifying this value.

NIMPS. The NIMPS command is used to specify the number of imputed data sets.

SEED. The SEED command is used to specify the pseudorandom number generator seeding value. The

input value to the command is required to be a positive integer with nine or fewer digits. Note that Blimp requires a seed value to run.

CHAINS. The CHAINS command is used to specify the number of MCMC chains. The default number of chains is one, unless the psr keyword is specified on the OPTIONS command, in which case two chains are used. When multiple chains are specified, Blimp attempts to distribute the chains across physical cores, resulting in faster computation. To override this default behavior, one can modify the number of processors with the processors keyword, as follows.

```
CHAINS: 10 processors 1;
```

The above line specifies 10 chains using only one processor at a time. It is important to note that each chain will have a different seeding value along with different randomly perturbed start values for the parameters in the imputation model.

OUTFILE. The OUTFILE command is used to specify the location and name of the imputed data set(s). The OUTFILE command requires the full file path to be specified. As explained in the OPTIONS section below, Blimp can stack the imputed data sets in a single file, or it can save each imputed data set to a separate file. The stacked format is compatible with most general-use software packages (e.g., SAS, SPSS, Stata), but some packages such as M*plus* require imputations as separate files. When saving imputations to separate files, an asterisk (*) is required in the file path (e.g., see Example 2.1 in the Quick Start Chapter of the document). This asterisk will be replaced with an integer in the file name (e.g., specifying myimps*.dat would produce imputed data sets named myimps1.dat, myimps2.dat, etc.). The asterisk convention is also required when employing Blimp's internal simulation features, which are described in Chapter 8. No asterisk is needed when writing data sets to a single stacked file.

OPTIONS. The OPTIONS command is used to specify algorithmic and other miscellaneous options (e.g., the format of the output data sets). If this command is excluded, Blimp will rely on default settings. Keywords (described below) are separated by a space (e.g., see Example 2.1 in the Quick Start Chapter).

OPTIONS Keywords

This section describes the keywords available with the OPTIONS command. Note that uppercase keywords in bold typeface are default settings that will be used unless otherwise specified.

MICE/GIBBS. Blimp offers two versions of the fully conditional specification algorithm. Each variable

is imputed one at a time from a regression model that features an incomplete variable regressed on all remaining variables listed on the `MODEL` command. A Bayesian estimation sequence provides the necessary parameter estimates. The `mice` keyword is consistent with the original formulation of fully conditional specification described by van Buuren et al. (2011), where estimates are derived using only those cases with observed data on the variable to be imputed. The `gibbs` keyword triggers the traditional Gibbs sampler described in Bayesian analysis texts whereby parameter estimates are derived from the full imputed data set. The `mice` option is the default, but the `gibbs` option can be used to avoid computational problems that occur when some clusters have few cases (e.g., singleton clusters) or very sparse data. With continuous variables, the two options give nearly identical results, although the `gibbs` option tends to converge more slowly. Note that substantive model-compatible imputation (SMC-FCS) requires a Metropolis sampler.

CSV/`DAT`. Blimp can save imputed data sets in comma separated values (CSV) format or in space-delimited format. The `csv` and `dat` keywords specify these options, respectively, with `csv` as the default.

STACKED/`SEPARATE`. The `stacked` keyword produces a single file with imputed data sets stacked one on top of another. In this case, the first variable in the output data set is an identifier variable that indexes the data sets. The `separate` keyword writes each imputed data set to a separate text file. This specification requires an asterisk in the filename, and this asterisk is subsequently replaced by a numeric value (e.g., a filename of myimps*.dat would produce imputed data sets named myimps1.dat, myimps2.dat, etc.). The choice of stacked or separate files depends on the analysis software being used. The stacked format is compatible with most general-use software packages (e.g., SAS, SPSS, Stata), but some packages such as M*plus* require imputations as separate files. To facilitate the use of Blimp data with M*plus*, the `separate` keyword also creates a list file that contains the file paths of the imputed data sets. This list file is used as the input data for an M*plus* analysis.

HOV/`HEV`. In a two-level data set, the `hov` (`homogenous`) keyword specifies a common residual variance for all clusters, which is in line with the standard representation of the multilevel model. The `hev` (`heterogeneous`) keyword specifies an imputation model with heterogeneous within-cluster residual variances (see Kasim & Raudenbush, 1998; van Buuren et al., 2011). Such a specification might be warranted with daily diary data, for example.

CLMEAN/`NOCLMEAN`. With multilevel data, the `clmean` keyword introduces the cluster means as additional predictors in the imputation model. For example, suppose that `x` and `y` are level-1 variables in a two-level imputation scheme. The `clmean` keyword introduces `x` and its cluster means as predictors

in the y imputation model, and vice versa. The clmean keyword is particularly important for analyses that posit unique within-cluster and between-cluster covariance structures. Such is the case with contextual effects regression models and many multilevel structural equation models (e.g., a multilevel factor model that posits different factor structures or factor loadings at level-1 and level-2). The noclmean keyword specifies a parsimonious imputation model that does not partition associations into their within- and between-cluster components. The noclmean option is appropriate for analysis models where the level-1 and level-2 regressions are assumed to be identical (i.e., a model with no contextual effects).

PRIOR1/PRIOR2. For all regression coefficients in the imputation model, Blimp implements standard non-informative prior distributions, and these priors cannot be modified. The program offers two choices of prior distribution for variance parameters. In a multilevel model, priors can be specified for level-2 (and level-3) covariance matrices and the within-cluster residual variance. In a single-level model, the residual variance is the only parameter to which the priors apply. The prior1 keyword implements an inverse Wishart, $W^{-1}(\mathbf{I}, p+1)$, where $\mathbf{I}$ is a $p \times p$ identity matrix and p is the number of variance estimates in the covariance matrix. The prior2 keyword specifies an inverse Wishart prior, $W^{-1}(0, -p-1)$, where p is the number of variance estimates in the covariance matrix. In a multilevel application, the prior1 keyword is the default, and prior2 is the default for single-level models. When using the OUTCOME command (i.e., SMC-FCS) prior2 will be the default. In Monte Carlo simulations, we have determined that these default priors tend to provide the best parameter recovery in a broad range of scenarios, but it may be advisable to examine whether the choice of prior materially impacts the subsequent analysis results.

NOPSR/PSR. The psr keyword prints a table of potential scale reduction (PSR) factors (Gelman & Rubin, 1992). The PSR values can be used to diagnose the convergence of the MCMC algorithm. Imputation for each incomplete variable is based on a regression model, the parameters of which define a distribution of plausible imputations. In a multilevel model, the parameters consist of regression coefficients, covariance matrices (level-2 or level-3), a within-cluster residual variance, and threshold parameters (ordinal variables with more than two categories). In a single-level model, the imputation model parameters are regression coefficients, a residual variance, and possibly threshold parameters. PSR values are computed for every parameter in the imputation models, and the psr keyword prints a brief summary table that reflects the worst (highest) PSR value for the regression coefficients, variance/covariance estimates, and threshold parameters (if applicable). The nopsr keyword is the default, meaning that no diagnostic information is printed. See Chapter 6 for more information.

4 Running Blimp

Blimp is executed via the 'Run' pull-down option from the GUI or from the command-line. Double-clicking a Blimp syntax file (a text file ending in the .imp extension) will automatically open the GUI. The syntax file is then submitted for execution by selecting the Run option from the 'Impute' pull-down menu. Alternatively, Blimp can be executed via the command-line. A terminal emulator program (e.g., Terminal.app in macOS, cmd.exe in Windows) is used to run Blimp from the command line. The command line arguments for Blimp provide flexibility for advanced applications, primarily external Monte Carlo computer simulations. This application is covered in Chapter 8, and this chapter will focus on running Blimp with a single data set.

The Windows version of Blimp automatically allows users to execute files via the command line. In macOS, the executable file is packaged in the application bundle and is not immediately accessible to users. To enable this functionality from the GUI, select the command line Tool option from the main Blimp pull-down menu. A pop-up dialog box will ask whether the command line tool should be installed. After enabling this functionality, Blimp can be executed from a shell program such as the Terminal.app application.

To run Blimp from a command line, type blimp followed by the full file path to the syntax file. For example, the command line specification for macOS is

```
$ blimp ~/desktop/InputExample.imp
```

where 'InputExample.imp' is a Blimp syntax script saved in the user's desktop directory. Most terminal applications (macOS or Windows) automatically generate the appropriate file paths when the application and syntax files are dragged and dropped into the terminal window.

5 Blimp Output

After executing Blimp (GUI or shell), a header is printed to the output window. This header provides the version number along with contact information. An example header is shown below.

```
                                Blimp
                                 v1.0

          Developed by Craig K. Enders and Brian T. Keller.
                Blimp was developed with funding from
            Institute of Educational Sciences award R305D150056.
            Craig K. Enders, P.I. Email: cenders@psych.ucla.edu.
         Programming by Brian T. Keller. Email: bkeller2@ucla.edu
                   There is no expressed license given.


                DO NOT DISTRIBUTE WITHOUT WRITTEN CONSENT
```

Below the header, Blimp lists the algorithmic options specified in the syntax file. In addition to specified options, the list includes default settings that may not have been explicitly specified in the syntax. For example, in Example 2.1 the keywords `hov` and `clmean` were not specified in the `OPTIONS` command, so these options were enabled by default. The syntax did not specify prior distributions for variance components, thus triggering appropriate default settings.

```
ALGORITHMIC OPTIONS SPECIFIED:

  Imputation method:                Fully conditional specification
  MCMC algorithm:                   MICE sampler (MICE)
  Between-cluster imputation model: Cluster means included (CLMEAN)
  Residual variance structure:      Homogeneous level-1 variance (HOV)
  Prior for covariance matrices:    Identity matrix, df = p + 1 (PRIOR1)
  Prior for residual variance:      Unit sum of squares, df = 2 (PRIOR1)
  Diagnostics:                      No potential scale reduction (NOPSR)
  Imputation format:                Separate files (SEPARATE)
```

Following the algorithmic options, Blimp displays a summary of the imputation model variables.

```
VARIABLES IN IMPUTATION MODEL:

  Level-2 identifier:     id
  Complete variables:     av x3#1 x3#2 x3#3
  Incomplete continuous:  y x1 w1
  Incomplete ordinal:     x2 w2
  Incomplete nominal:     w3
```

Next, the output displays the progress of the MCMC algorithm. Example 2.1 requests 2000 burn-in iterations, and the program issues a message when it has completed 25%, 50%, 75%, and 100% of these initial iterations. Following the burn-in phase, Blimp prints a time and date stamp as it saves imputed data sets.

```
ITERATION HISTORY:

Starting Burn-in on Fri Sep  8 10:07:35 2017
    Burn-in iteration 500 complete on Fri Sep  8 10:07:37 2017
    Burn-in iteration 1000 complete on Fri Sep  8 10:07:40 2017
    Burn-in iteration 1500 complete on Fri Sep  8 10:07:43 2017
    Burn-in iteration 2000 complete on Fri Sep  8 10:07:46 2017
Burn-in complete on Fri Sep  8 10:07:46 2017
    Imputation Saved 1 on Fri Sep  8 10:07:46 2017
    Imputation Saved 2 on Fri Sep  8 10:07:49 2017
    Imputation Saved 3 on Fri Sep  8 10:07:53 2017
    Imputation Saved 4 on Fri Sep  8 10:07:56 2017
    Imputation Saved 5 on Fri Sep  8 10:08:00 2017
    Imputation Saved 6 on Fri Sep  8 10:08:03 2017
    Imputation Saved 7 on Fri Sep  8 10:08:08 2017
    Imputation Saved 8 on Fri Sep  8 10:08:12 2017
    Imputation Saved 9 on Fri Sep  8 10:08:16 2017
    Imputation Saved 10 on Fri Sep  8 10:08:20 2017
```

Finally, Blimp prints the order of the variables in the imputed data set(s). All variables listed on the VARIABLES command are saved to the file(s), regardless of whether the variables are used in imputation.

```
VARIABLE ORDER IN SAVED DATA:

   id av y x1 x2 x3 w1 w2 w3
```

Variables are saved in the same order as they appear on the VARIABLES command, with one important exception. When imputations are stacked in a single file (the stacked keyword of the OPTION command), the

first variable in the file is an identifier variable that indexes the imputed data sets. In this case, the variable order appears as follows.

```
VARIABLE ORDER IN SAVED DATA:

  imp# id av y x1 x2 x3 w1 w2 w3
```

As noted previously, product or polynomial terms specified on the `MODEL` command do not appear in the output data files. These variables must be computed prior to analysis. For examples, see the analysis scripts available at www.appliedmissingdata.com. Finally, the Blimp output will print the PSR summary tables if the `psr` keyword is specified on the `OPTIONS` command. These tables are discussed in Chapter 6.

Multiple Chains

When multiple chains are requested, Blimp prints slightly different output. The follow excerpts of output are based on an example of eight chains and eight imputations. Initially, each chain will display the seed value used. These seeds are randomly generated based on the seed value supplied in the `SEED` command.

```
Chain 1 online with seed of 158875
Chain 2 online with seed of 150835
Chain 3 online with seed of 98783
Chain 4 online with seed of 104168
```

During this time, no output will be displayed about the status of burn-in iterations. Instead, when a chain saves an imputation the chain number and the imputation number will be printed. It is important to note that the imputations may not finish in consecutive order.

```
    Chain 3 saving imputation 3 complete on Fri Sep  8 12:49:10 2017
    Chain 2 saving imputation 2 complete on Fri Sep  8 12:49:10 2017
    Chain 4 saving imputation 4 complete on Fri Sep  8 12:49:10 2017
    Chain 1 saving imputation 1 complete on Fri Sep  8 12:49:10 2017
```

Upon completion of all imputations required in the first set of chains, the next set of chains will start and subsequently save imputations. This process will continue until all chains and imputations have completed.

```
Chain 5 online with seed of 172863
Chain 6 online with seed of 55365
Chain 7 online with seed of 32484
Chain 8 online with seed of 34529
    Chain 7 saving imputation 7 complete on Fri Sep  8 12:49:10 2017
    Chain 5 saving imputation 5 complete on Fri Sep  8 12:49:10 2017
    Chain 6 saving imputation 6 complete on Fri Sep  8 12:49:10 2017
    Chain 8 saving imputation 8 complete on Fri Sep  8 12:49:10 2017
```

6 Convergence Diagnostics with Blimp

The `psr` keyword of the `OPTIONS` command prints a table of potential scale reduction (PSR) factors (Gelman & Rubin, 1992). The PSR values can be used to diagnose the convergence of the iterative MCMC algorithm. PSR values less than 1.05 to 1.10 are typically viewed as acceptable. Unless otherwise specified, specifying the `psr` keyword invokes two MCMC chains. PSR values are calculated after every 100 iterations of the specified burn-in period. Each time the PSR is computed, the first half of the iterations are discarded, and PSR values are computed by comparing parameter estimates from the two chains. Note that Blimp will compute the PSR factors only for the burn-in period. Therefore, when the burn-in period is finished, Blimp will continue to run until the desired number of imputed data sets are obtained and print the PSR factors for the burn-in iterations afterwards.

Imputation for each incomplete variable is based on a regression model, the parameters of which define a distribution of plausible imputations. In a multilevel model, the parameters consist of regression coefficients, between-cluster variance/covariance matrices (level-2 and level-3), a within-cluster residual variance, and threshold parameters (ordinal variables with more than two categories). In a single-level model, the imputation model parameters include regression coefficients, a residual variance, and possibly threshold parameters. PSR values are computed for every parameter in the imputation models, and the output prints summary tables that reflect the highest (worst) PSR value for the regression coefficients, variance/covariance estimates, and threshold parameters (if applicable).

The tabled values of the PSR are divided into four categories: Fixed effects regression coefficients (labeled `Fix Eff`), between-cluster variance-covariance matrix parameters (labeled `Ran Eff Var`), within-cluster residual variance parameters (labeled `Err Var`), and threshold parameters for ordinal variables with more than two categories (labeled `Threshold`). Below each category label Blimp displays the maximum PSR across all imputation models for a given parameter category. An example PSR table is shown below.

```
POTENTIAL SCALE REDUCTION (PSR) OUTPUT:

  Comparing iterations 51 to 100 for 2 chains.
                     ---------------------------------------------------
                     |     Fix Eff| Ran Eff Var|     Err Var|   Threshold|
                     ---------------------------------------------------
            Max PSR  |       1.434|       1.351|       1.081|       1.043|
   Missing Variable  |          x1|          x1|           y|          w2|
                     ---------------------------------------------------
```

The table above reflects the PSR computations from a two-chain MCMC process at the 100th iteration of the burn-in phase. The first 50 iterations are discarded, and PSR values are computed by comparing parameter estimates from two separate MCMC chains with 50 iterations (iterations 51 through 100). As a second example, consider the PSR calculation at iteration 600. Again, the first half of the iterations are discarded, and PSR values are computed by comparing parameter estimates from two separate MCMC chains with 300 iterations (iterations 301 through 600). An example table is shown below.

```
  Comparing iterations 301 to 600 for 2 chains.
                     ---------------------------------------------------
                     |     Fix Eff| Ran Eff Var|     Err Var|   Threshold|
                     ---------------------------------------------------
            Max PSR  |       1.023|       1.041|       1.005|       1.007|
   Missing Variable  |          x2|           y|           y|          w2|
                     ---------------------------------------------------
```

As noted above, PSR values are computed for every parameter in the imputation models, and the output prints summary tables that reflect the worst (highest) PSR value for the regression coefficients, variance/covariance estimates (if applicable), and threshold parameters (if applicable). The labels in the Variable row reflect the incomplete variable that generated each PSR value. For example, in the table above, the highest PSR value across all regression coefficients is associated with the `x2` imputation model, and the highest PSR for any between-cluster variance component is from the `y` imputation model. As noted previously, PSR values less than 1.05 to 1.10 are typically viewed as acceptable. Threshold parameters tend to converge very slowly, and we have observed no negative consequences from adopting a slightly less stringent criterion for these parameters (e.g., PSR values below 1.10 to 1.15).

7 Analysis Examples

This chapter illustrates the application of Blimp to a variety of real-world analysis problems. The examples are inspired by an Institute of Educational Sciences-funded project featuring a cluster-randomized trial of a novel math problem-solving intervention (Montague, Krawec, Enders, & Dietz, 2014). The data structure consists of three levels: repeated measurements at level-1 nested in students at level-2, and students in turn nested in schools at level-3. Schools were randomly assigned to an intervention or control condition, such that all students within a given school received the same treatment. The analysis examples in this chapter use different configurations of the data to illustrate imputation for single-level and multilevel regression analyses. Artificial data sets based on the published data are available at www.appliedmissingdata.com, as are analysis scripts for M*plus*, SAS, SPSS, Stata, and R.

Examples 7.1 through 7.5 use a two-level version of the data set where students at level-1 are nested in schools at level-2. The variables in the file are as follows:

```
SCHOOL: School identifier variable
CONDITION: Intervention code (0 = control, 1 = intervention)
ESOLPERCENT: Percentage of students for whom English is a second language
STUDENT: Student identifier variable
ABILITYGRP: Three-group diagnostic classification (nominal)
FEMALE: Gender dummy code (0 = male, 1 = female)
STANMATH: Standardized math achievement scores
FRLUNCH: Lunch assistance code (0 = no, 1 = free/reduced lunch)
EFFICACY: Ordinal rating of math self-efficacy
PROBSOLVE1: Baseline problem-solving
PROBSOLVE7: End-of-year problem-solving
```

With the exception of gender, all variables have missing values, the code for which is 999.

Examples 7.6 and 7.7 use a three-level data set with repeated measures (level-1) nested in students (level-2) and students nested in schools (level-3). The variables in the file are as follows:

```
SCHOOL: School identifier variable
CONDITION: Intervention code (0 = control, 1 = intervention)
ESOLPERCENT: Percentage of students for whom English is a second language
STUDENT: Student identifier variable
ABILITYGRP: Three-group diagnostic classification (nominal)
FEMALE: Gender dummy code (0 = male, 1 = female)
STANMATH: Standardized math achievement scores
FRLUNCH: Lunch assistance code (0 = no, 1 = free/reduced lunch)
WAVE: Integer index for repeated measures
MONTHS: Months since start of the school year
PROBSOLVE: Math problem-solving
EFFICACY: Ordinal rating of math self-efficacy
```

With the exception of gender, data collection wave, and months since baseline, all variables have missing values, the code for which is 999.

Example 7.1: Single-Level Regression Analysis

Example 7.1 is a single-level regression analysis where baseline scores and student-level covariates predict end-of-year problem-solving scores. The substantive analysis model is as follows.

$$probsolve7_i = \gamma_0 + \gamma_1 \left(probsolve1_i\right) + \gamma_2 \left(efficacy_i\right) + \gamma_3 \left(female_i\right)$$
$$+ \gamma_4 \left(abilitygrp2_i\right) + \gamma_5 \left(abilitygrp3_i\right) + \epsilon_i$$

Note that the diagnostic classification variable is represented as two dummy codes. The Blimp scripts below include the analysis variables as well as standardized math scores and the lunch assistance indicator as dummy variables.

Diagnosing Convergence for Example 7.1

It is good practice to perform a diagnostic run to evaluate the convergence behavior of the MCMC algorithm. The Blimp script below specifies a 3000-cycle burn-in period, during which it computes the potential scale reduction (PSR) factors at regular intervals in the chain. The imputation model includes a mixture of continuous (normal) and categorical variables, and such combinations often require many iterations to converge.

Blimp computes the PSR factors in 100 iteration increments. Comparisons are made between two chains, with the first half of each chain discarded. The PSR tables from the Blimp output file suggest that two chains comprised of 600 iterates (iterations 601 to 1200, with iterations 1 through 600 discarded) produce

PSR values below 1.05, a value often deemed as acceptable for practice. Based on this information, the final imputation runs specify a conservative burn-in and thinning interval of 1000 iterations.

Example 7.1: Diagnostics Script

```
DATA: /Users/name/Desktop/exampledata1.csv;
VARIABLES: student abilitygrp female stanmath frlunch efficacy
   probsolve1 probsolve7;
ORDINAL: efficacy;
NOMINAL: abilitygrp female frlunch;
MISSING: 999;
MODEL: ~  abilitygrp female stanmath frlunch efficacy
   probsolve1 probsolve7;
CHAINS: 2 processors 2;
NIMPS: 2;
BURN: 3000;
THIN: 1;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: psr;
```

Example 7.1: Diagnostics Script PSR Output

```
POTENTIAL SCALE REDUCTION (PSR) OUTPUT:

  Comparing iterations 51 to 100 for 2 chains.
                     ------------------------------------------------------
                     |     Fix Eff| Ran Eff Var|     Err Var|   Threshold|
                     ------------------------------------------------------
            Max PSR  |       1.653|         nan|       1.006|       2.545|
   Missing Variable  |  abilitygrp|            |  probsolve7|    efficacy|
                     ------------------------------------------------------
```

⋮

```
  Comparing iterations 601 to 1200 for 2 chains.
                     ------------------------------------------------------
                     |     Fix Eff| Ran Eff Var|     Err Var|   Threshold|
                     ------------------------------------------------------
            Max PSR  |       1.024|         nan|       1.001|       1.043|
   Missing Variable  |  abilitygrp|            |  probsolve7|    efficacy|
                     ------------------------------------------------------
```

Separate Format for R, SAS, SPSS, and Stata Analysis

The Blimp script below produces 20 imputations that are saved to separate files name imp1.csv, imp2.csv, . . . , imp20.csv. In addition, Blimp creates a file called implist.csv that contains the names of these files. This additional file serves as the input data when analyzing the imputations in M*plus*.

Example 7.1: Separate Imputation Script

```
DATA: /Users/name/Desktop/exampledata1.csv;
VARIABLES: student abilitygrp female stanmath frlunch efficacy
   probsolve1 probsolve7;
ORDINAL: efficacy;
NOMINAL: abilitygrp female frlunch;
MISSING: 999;
MODEL: ~  abilitygrp female stanmath frlunch efficacy
   probsolve1 probsolve7;
NIMPS: 20;
BURN: 1000;
THIN: 1000;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imp*.csv;
OPTIONS: separate;
```

Requesting separate format does not add a new variable to the beginning of each file; therefore, the variable order maintains the same as the original data set.

```
VARIABLE ORDER IN SAVED DATA:

  student abilitygrp female stanmath frlunch efficacy probsolve1
    probsolve7
```

Example 7.2: Single-Level Regression with Interaction Effect

Example 7.2 is a single-level regression analysis that features a gender by self-efficacy interaction. The substantive analysis model is as follows.

$$probsolve7_i = \gamma_0 + \gamma_1 (probsolve1_i) + \gamma_2 (efficacy_i) + \gamma_3 (female_i) + \gamma_4 (efficacy_i)(female_i) + \epsilon_i$$

Because gender is complete, either SMC-FCS or separate-group (`BYGROUP`) imputation can be used to preserve any gender-specific relations that may exist in the data. The subsequent Blimp scripts illustrate these procedures. Because SMC-FCS is currently limited to continuous and ordinal (including binary) variables,

the nominal diagnostic classification variable cannot be included as an auxiliary variable. Standard FCS with BYGROUP imputation can include this variable, however. Both scripts produce a stacked data file. Stacked data format is useful when analyzing the data in R, SAS, SPSS, and Stata. Requesting stacked format adds a new variable to the beginning of each file that indexes the imputations, as shown near the bottom of the Blimp output. The burn-in and thinning intervals were specified after examining the convergence diagnostics illustrated in Example 7.1.

Example 7.2: Substantive Model-Compatible FCS Script

```
DATA: /Users/name/Desktop/exampledata1.csv;
VARIABLES: student abilitygrp female stanmath frlunch
   efficacy probsolve1 probsolve7;
ORDINAL: female frlunch efficacy;
OUTCOME: probsolve7;
MISSING: 999;
MODEL: ~  female stanmath frlunch efficacy
   probsolve1 probsolve7 female*efficacy;
NIMPS: 20;
BURN: 1000;
THIN: 1000;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked;
```

Example 7.2: Standard FCS with Separate-Group (BYGROUP) Imputation Script

```
DATA: /Users/name/Desktop/exampledata1.csv;
VARIABLES: student abilitygrp female stanmath frlunch
   efficacy probsolve1 probsolve7;
ORDINAL: efficacy;
NOMINAL: abilitygrp frlunch;
BYGROUP: female;
MISSING: 999;
MODEL: ~  abilitygrp stanmath frlunch
   efficacy probsolve1 probsolve7;
NIMPS: 20;
BURN: 1000;
THIN: 1000;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked;
```

Example 7.3: Two-Level Regression with Random Intercepts

Example 7.3 is a two-level regression analysis with random intercepts, where intervention condition and covariates predict end-of-year problem-solving scores. The substantive analysis model is as follows.

$$probsolve7_{ij} = \gamma_0 + \gamma_1\left(probsolve1_{ij}\right) + \gamma_2\left(efficacy_{ij}\right) + \gamma_3\left(female_{ij}\right) + \gamma_4\left(abilitygrp2_{ij}\right) + \gamma_5\left(abilitygrp3_{ij}\right) + \gamma_6\left(esolpercent_j\right) + \gamma_7\left(condition_j\right) + u_{0_j} + \epsilon_{ij}$$

Consistent with the previous examples, the diagnostic classification variable is represented as two dummy codes. The Blimp script below produces a stacked data file, and the burn-in and thinning intervals were specified after examining the convergence diagnostics illustrated in Example 7.1.

Example 7.3: Two-level Regression with Random Intercept Imputation Script

```
DATA: /Users/name/Desktop/exampledata2.csv;
VARIABLES: school condition esolpercent student abilitygrp
   female stanmath frlunch efficacy probsolve1 probsolve7;
ORDINAL: efficacy;
NOMINAL: abilitygrp condition female frlunch;
MISSING: 999;
MODEL: school ~  condition esolpercent abilitygrp female
   stanmath frlunch efficacy probsolve1 probsolve7;
NIMPS: 20;
BURN: 1500;
THIN: 1500;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked;
```

Example 7.4: Two-Level Regression with Random Slopes

Example 7.4 is a two-level regression analysis where intervention condition and covariates predict end-of-year problem-solving scores, and self-efficacy is specified as a random coefficient. The substantive analysis model is as follows.

$$probsolve7_{ij} = \gamma_0 + \gamma_1\left(probsolve1_{ij}\right) + \gamma_2\left(efficacy_{ij}\right) + \gamma_3\left(female_{ij}\right) + \gamma_4\left(esolpercent_j\right) + \gamma_5\left(condition_j\right) + u_{0_j} + u_{1_j}\left(efficacy_{ij}\right) + \epsilon_{ij}$$

When possible, SMC-FCS is preferred for models with random slopes, as this procedure generally yields more accurate between-cluster variance estimates. The following Blimp scripts illustrate both SMC-FCS

and standard FCS with so-called "reverse random coefficients." The Blimp scripts below produce a stacked data file, and the burn-in and thinning intervals were specified after examining the convergence diagnostics illustrated in Example 7.1.

Example 7.4: Substantive Model-Compatible FCS Imputation Script

```
DATA: /Users/name/Desktop/exampledata2.csv;
VARIABLES: school condition esolpercent student abilitygrp
   female stanmath frlunch efficacy probsolve1 probsolve7;
ORDINAL: condition female frlunch efficacy;
OUTCOME: probsolve7;
MISSING: 999;
MODEL: school ~  condition esolpercent female stanmath
   frlunch probsolve1 efficacy:probsolve7;
NIMPS: 20;
BURN: 1500;
THIN: 1500;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked;
```

Example 7.4: Standard FCS with "Reverse Random Coefficients" Script

```
DATA: /Users/name/Desktop/exampledata2.csv;
VARIABLES: school condition esolpercent student abilitygrp
   female stanmath frlunch efficacy probsolve1 probsolve7;
ORDINAL: condition female frlunch efficacy;
MISSING: 999;
MODEL: school ~  condition esolpercent female stanmath
   frlunch probsolve1 efficacy:probsolve7;
NIMPS: 20;
BURN: 1500;
THIN: 1500;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked;
```

Example 7.5: Two-Level Regression with Random Slopes and a Cross-Level Interaction Effect

Example 5 is a two-level regression analysis that features a random slope and a cross-level interaction involving condition and self-efficacy ratings. The substantive analysis model is as follows.

$$probsolve7_{ij} = \gamma_0 + \gamma_1 \left(probsolve1_{ij}\right) + \gamma_2 \left(efficacy_{ij}\right) + \gamma_3 \left(female_{ij}\right) + \gamma_4 \left(esolpercent_j\right) + \gamma_5 \left(condition_j\right) + \gamma_6 \left(efficacy_{ij}\right) \left(condition_j\right) + u_{0_j} + u_{1_j} \left(efficacy_{ij}\right) + \epsilon_{ij}$$

Because the intervention code is complete, either SMC-FCS or separate-group (`BYGROUP`) FCS imputation can be used to preserve the interaction. However, SMC-FCS is preferable for models with random slopes, and so the subsequent Blimp script illustrates this procedure. The burn-in and thinning intervals were specified after examining the convergence diagnostics illustrated in Example 7.1.

Example 7.5: Two-Level Regression with Random Slopes and a Cross-Level Interaction Imputation Script

```
DATA: /Users/name/Desktop/exampledata2.csv;
VARIABLES: school condition esolpercent student abilitygrp
   female stanmath frlunch efficacy probsolve1 probsolve7;
ORDINAL: condition female frlunch efficacy;
OUTCOME: probsolve7;
MISSING: 999;
MODEL: school ~  condition esolpercent female stanmath
   frlunch probsolve1 efficacy:probsolve7 efficacy*condition;
NIMPS: 20;
BURN: 1500;
THIN: 1500;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked;
```

Example 7.6: Three-Level Regression with Random Intercepts

Example 7.6 is a three-level regression analysis with random intercepts, where intervention condition and covariates predict problem-solving scores. The substantive analysis model is as follows.

$$probsolve7_{ijk} = \gamma_0 + \gamma_1 \left(efficacy_{ijk}\right) + \gamma_2 \left(female_{jk}\right) + \gamma_3 \left(abilitygrp2_{jk}\right) + \gamma_4 \left(abilitygrp3_{jk}\right) + \gamma_5 \left(esolpercent_k\right) + \gamma_6 \left(condition_k\right) + u_{0_{jk}} + u_{0_k} + \epsilon_{ijk}$$

Consistent with the previous examples, the diagnostic classification variable is represented as two dummy codes. The Blimp script below produces a stacked data file, and the burn-in and thinning intervals were specified after examining the convergence diagnostics illustrated in Example 7.1.

Example 7.6: Three-Level Regression with Random Intercepts Imputation Script

```
DATA: /Users/name/Desktop/exampledata3.csv;
VARIABLES: school condition esolpercent student abilitygrp
   female stanmath frlunch wave months probsolve efficacy;
ORDINAL: condition female frlunch efficacy;
NOMINAL: abilitygrp;
MISSING: 999;
MODEL: student school ~  condition esolpercent abilitygrp
   female stanmath frlunch probsolve efficacy;
NIMPS: 20;
BURN: 2000;
THIN: 2000;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked gibbs;
```

Example 7.7: Three-Level Regression with Random Slopes and Cross-Level Interaction Effect

Example 7.7 is a three-level growth curve analysis that features a random slope for the "time" variable (months since the beginning of the school year) and a cross-level interaction involving condition and months (i.e., the group-by-time interaction). The substantive analysis model is as follows.

$$\begin{aligned} probsolve7_{ijk} = {} & \gamma_0 + \gamma_1 \left(efficacy_{ijk}\right) + \gamma_2 \left(months_{ijk}\right) + \gamma_3 \left(female_{jk}\right) + \gamma_4 \left(esolpercent_k\right) \\ & + \gamma_5 \left(condition_k\right) + \gamma_6 \left(months_{ijk}\right) \left(condition_k\right) + u_{0_{jk}} + u_{1_{jk}} \left(months_{ijk}\right) \\ & + u_{0_k} + u_{1_k} \left(months_{ijk}\right) + \epsilon_{ijk} \end{aligned}$$

Separate-group (`BYGROUP`) processing within standard FCS imputation can be used to preserve interaction effects when one of the component variables is categorical and complete, which it is here. Because the random "time" predictor is also complete, this procedure would work well, but it would require a relatively complex covariance structure that modeled unique random slopes for each group. In contrast, SMC-FCS does not require many group-specific parameters and thus is more parsimonious. The following Blimp script illustrates SMC-FCS, and the burn-in and thinning intervals were specified after examining the convergence diagnostics

illustrated in Example 7.1.

Example 7.7: Three-Level Regression with Random Slopes and Cross-Level Interaction Imputation Script

```
DATA: /Users/name/Desktop/exampledata3.csv;
VARIABLES: school condition esolpercent student abilitygrp
   female stanmath frlunch wave months probsolve efficacy;
ORDINAL: condition female frlunch efficacy;
OUTCOME: probsolve;
MISSING: 999;
MODEL: student school ~  condition esolpercent
   female stanmath frlunch months:probsolve efficacy
   months*condition;
NIMPS: 20;
BURN: 3000;
THIN: 3000;
SEED: 90291;
OUTFILE: /Users/name/Desktop/imps.csv;
OPTIONS: stacked;
```

8 Running a Simulation

Blimp has utilities to run simulations with two different methods: (a) internal Blimp simulations and (b) external Blimp simulations. Internal simulations use Blimp to automate the imputation of multiple replicates with a single Blimp syntax file. External Blimp simulations allow users to impute replicates with additional scripting from outside programs (e.g., R, Bash, etc.). Both methods are designed to easily automate Blimp for the purposes of methodological simulations.

Internal Blimp Simulations

Internal Blimp simulations offer users an easy interface to quickly run Blimp on a collection of simulated data sets. Example 8.1 below demonstrates an internal simulation. For an internal simulation, a set of artificial data sets must have a common file name and a consecutive integer ranging from one to the desired number of replications (e.g., myrep1.dat, myrep2.dat, ..., myrep1000.dat). The `SIMULATE` command is added to the first line of the Blimp syntax file, and the command has two keywords. The `replications` keyword must be followed by a space and the number of artificial data sets. Blimp will start from one and increment the number by one in the file path until reaching the total number of replications. In addition, a range of replications can be executed by specifying a starting replication, the keyword `to` and the ending replication. For example, `replications 200 to 500`. The `processors` keyword must be followed by a space and the number of total processors used. Blimp will restrict the number of processors based on computer hardware specifications (and allow no more than eight total). Specifying multiple processors will run each replication on a separate processor. For example, specifying: `processors 4` will run 4 replications at a time until all are finished.

Example 8.1: Internal Simulation Syntax

```
SIMULATE: replications 1000 processors 4;

DATA: /Users/name/Desktop/myrep*.dat;

VARIABLES: idvar x y z;

MODEL: idvar ~ x y z;

BURN: 500;

THIN: 100;

NIMPS: 10;

MISSING: -9999;

SEED: 38203;

CHAINS: 1;

OUTFILE: /Users/name/Desktop/myimps*.csv;

OPTIONS: csv;
```

When invoking the `SIMULATE` command, the `DATA` command is used to specify the file name prefix for the artificial data sets, and an asterisk is used as a placeholder for the replication number. To illustrate, consider Example 8.1. The `SIMULATE` command specifies 1000 replications, and the `DATA` command indicates that the file names are myrep1.dat, myrep2.dat, ..., myrep1000.dat. The `OUTFILE` command follows a similar convention. In Example 8.1, the imputed data sets are saved to files named myimps1.csv, myimps2.csv, ..., myimps1000.csv. Note that one cannot specify the `separate` keyword when using the `SIMULATE` command, as the imputed data sets for each replication will always be saved in the stacked format described in Chapter 3. In addition, the `psr` keyword is not available with the `SIMULATE` command.

Output from **SIMULATE** command

The SIMULATE command produces the same screen output if it is a single processor simulation or a multiprocessor simulation. Adjusting the input in Example 8.1 to run five replications would produce the following output after Blimp's standard header.

```
WARNING: Entering simulation mode.

ALGORITHMIC OPTIONS SPECIFIED:

  Imputation method:                 Fully conditional specification
  MCMC algorithm:                    MICE sampler (MICE)
  Between-cluster imputation model:  Cluster means included (CLMEAN)
  Residual variance structure:       Homogeneous level-1 variance (HOV)
  Prior for covariance matrices:     Identity matrix, df = p + 1 (PRIOR1)
  Prior for residual variance:       Unit sum of squares, df = 2 (PRIOR1)
  Diagnostics:                       No potential scale reduction (NOPSR)
  Imputation format:                 Stacked file (STACKED)

-------------------------------------------------------------------------

REPLICATION HISTORY:

Starting simulation on Fri Sep  8 12:56:05 2017
   Finished replication 1 on Fri Sep  8 12:56:05 2017
   Finished replication 4 on Fri Sep  8 12:56:05 2017
   Finished replication 2 on Fri Sep  8 12:56:05 2017
   Finished replication 3 on Fri Sep  8 12:56:05 2017
   Finished replication 5 on Fri Sep  8 12:56:05 2017
Finished simulation on Fri Sep  8 12:56:05 2017

-----------------------------------------------------------------

VARIABLE ORDER IN SAVED DATA:

  imp# x y z
```

Note that the replications will not necessarily finish in numerical order when more than one processor is used. Also note that if any errors occur that force a replication to exit, the replication will not have any imputations saved. That is, if the MCMC algorithm encounters computational problems for a particular replication after the 1st imputation is complete, the program does not produce an output data set that contains a partial set of imputations. Instead, Blimp will print out an error message that lists the failed replication number. Convergence problems can often be solved by specifying a different seed for the failed replicate, and failed replications can be addressed with the external simulation procedure described below.

External Blimp Simulations

An external Blimp simulation uses a common Blimp syntax file but changes specific commands in that script via command line arguments. The use of external simulations allows advanced users to circumvent some of the limiting factors of internal simulations (e.g., inability to run diagnostics, replication numbers must increment by one, etc.). The command line arguments for an external Blimp simulation are demonstrated by the following terminal commands.

```
$ /path/to/blimp/Blimp /path/to/input/InputExample.imp
       -d /path/to/data/data1.dat
       -o /path/to/output/output1.dat
       -s 287123
```

As before, the terminal command requires two file paths, the path to the executable, and the path to the Blimp syntax file. In the above example, three command-line arguments modify the file "InputExample.imp" (e.g., the script file that will be applied to a number of artificial data sets).

First, the `-d` or `--data` argument will override the file path given by the `DATA` command in the syntax file. A warning is displayed when this argument is used.

```
WARNING: Overriding DATA command in input.
```

The `-o` or `--outfile` argument will override the file path given by the `OUTFILE` command in the syntax file. Again, a warning is displayed when this argument is used.

```
WARNING: Overriding OUTFILE command in input.
```

The `-s` or `--seed` argument will override the `SEED` command in the syntax file. A warning is displayed when this argument is used.

```
WARNING: Overriding SEED command in input.
```

In order to automate a simulation, a scripting or programming language must be employed to submit the command-line arguments to the operating system. For convenience, we have included a script to do this in Example 8.2 using the R statistical programming language the `system()` function.

Example 8.2: R script to automate simulation external of Blimp

```
## Running a simulation via External Blimp method.
#    Note no spaces should be in the directory paths
#    All output from Blimp Is saved in a list: outputs
#    Does not work with 'sep' subcommand

# Set a Seed
seed       <- 37298372

# Path to Blimp
blimpPath  <- '~/Desktop/Blimp'

# Specify Path to Syntax File
inputPath  <- '~/Desktop/myInput.imp'

# Specify Path to Data Folder
#   Only the data files should be in folder.
dataPath   <- '~/Desktop/myDataFolder'

# Specify Path to Output Folder
outputPath <- '~/Desktop/myImpsFolder'

# Specify Names of Imputation Data
#   Use a * to represent where the name of data file.
#   E.g., Data1.csv will give you impData1.csv
impsData   <- 'imp*.csv'

#########################################################
## PROGRAM BEGINS HERE
# Get file names
dataFiles <- list.files(path = dataPath)
dataFilesPath <-list.files(path = dataPath, full.names = T)
# Calculate total number of reps.
repNumber <- length(dataFiles)
# Set seed
set.seed(seed)
# Generate list of seeds
seeds <- sample.int(1e10, repNumber,replace = F)
# Execute Simulation
outputs <- lapply(seq_along(seeds),function(x){
  # Construct path names
  repla <- gsub('\\..+','',dataFiles[x],perl=T)
  outfile <- paste0(outputPath,gsub('\\*',repla,impsData,perl=T))
  out <- system(paste(blimpPath,inputPath,'-o',outfile,'-s',
                seeds[x],'-d',dataFilesPath[x]),intern = T)
  return(list(dataFiles[x],out))
})
```

9 Error and Warning Message Reference

Error messages (denoted with ERROR in the Blimp output) are issued when problems arise due to computational or syntactical issues and cause Blimp to stop running and exit. In contrast, warning messages (denoted with WARNING in the Blimp output) will only warn the user that some options may have been adjusted, but Blimp will continue to run. The following sections detail Blimp error and warning messages and possible resolutions for each error.

Error Messages

```
ERROR: Unable to open syntax file.
```

Blimp was unable to open the Blimp syntax file. This usually indicates that the file path supplied to the Blimp cannot be found.

```
ERROR: Unable to expand syntax. The prefixes for
       c1 and x3 do not match.
```

A variable list that uses a dash to indicate a sequence of variables on the VARIABLE, NOMINAL, ORDINAL, or MODEL commands is incorrectly specified, most likely because the alphanumeric prefixes provided do not match. In the example above, the variable list c1-x3 would generate the error.

```
ERROR: Unable to expand syntax. The suffixes for
        x1 and x3a do not match.
```

A variable list that uses a dash to indicate a sequence of variables on the VARIABLE, NOMINAL, ORDINAL, or MODEL commands is incorrectly specified, most likely because the numeric suffixes provided do not match. In the example above, the variable list x1-x3a would generate the error.

```
ERROR: Must specify SIMULATE command before specifying data command.
```

The SIMULATE command must be specified before the DATA command in the Blimp syntax file. For more information on how to properly set up a simulation input see Chapter 8.

```
ERROR: Must specify more than 0 replications.
```

Blimp read the number of replications specified in the SIMULATE command as 0. Check input to verify that the number is greater than 0. For more information on how to properly set up a simulation input see Chapter 8.

```
ERROR: Must specify more than 0 processors.
```

Blimp read the number of processors to use in the SIMULATE command as 0. Check input to verify that the number is greater than 0. For more information on how to properly set up a simulation input see Chapter 8.

```
ERROR: Chains commmand not currently available in simulation mode.
```

The CHAINS command was used with simulation mode. Only single chains are allowed with internal simulations. To specify multiple chains in a simulation you must run an external simulation. For more information on external simulations see Chapter 8.

```
ERROR: Simulation mode already specified, psr command not available.
```

Currently the psr keyword is not available in simulation mode. To obtain PSR factors in a simulation you must run an external simulation. For more information on external simulations see Chapter 8.

```
ERROR: Missing a DATA, VARIABLES, or MODEL command.
```

Either the DATA, VARIABLES, and/or MODEL are/is missing or not read correctly by Blimp. Double check that all lines end in a semicolon (;). For more information on specifying a Blimp syntax file see Chapter 3.

```
ERROR: Missing imputation parameters.
```

Either the BURN, THIN, MISSING, SEED, and/or NIMPS command(s) are/is missing or not read correctly by Blimp. Double check that all lines end in a semicolon (;). For more information on specifying a Blimp syntax file see Chapter 3.

```
ERROR: No output file given.
```

The OUTFILE command is missing or not read correctly by Blimp. Double check that all lines end in a semicolon (;). For more information on specifying a Blimp syntax file see Chapter 3.

```
ERROR: Please place ONE asterisk (*) in OUTFILE command.
```

Blimp did not detect an asterisk (*) in the OUTFILE command or Blimp detected more than one asterisk. This is required when the SIMULATE command is used or the separate keyword is specified in the OPTIONS command. For more information on specifying a Blimp syntax file see Chapter 3.

```
ERROR: More processors than simulation files requested.
```

More processors were requested than the number of replications specified in the SIMULATE command. For more information on setting up a simulation see Chapter 8.

```
ERROR: More processors than chains requested.
```

More processors were requested than the number of chains specified in the CHAINS command. For more information on specifying a Blimp syntax file see Chapter 3.

```
ERROR: More chains than imputations requested.
```

More chains than the number of imputations were requested in the syntax file. For more information on specifying a Blimp syntax file see Chapter 3.

```
ERROR: Requested seed of --- must be greater than 0.
```

All seeds must be positive integers.

```
ERROR: DATA command and OUTFILE command have the same file path.
```

The file path specified in the DATA command and in the OUTFILE command are the same. These two paths must differ.

```
ERROR: File /MY/FILE/PATH cannot be found.
```

The file specified in the DATA command (labeled as '/MY/FILE/PATH' in the example error message) does not exist. Check to see if the line is terminated by a semicolon (;).

```
ERROR: File /MY/FILE/PATH cannot be created.
       Please check that the file path is correct.
```

The file specified in the OUTFILE command (labeled as '/MY/FILE/PATH' in the example error message) cannot be created. Check to see if the line is terminated by a semicolon (;).

```
ERROR: Currently no more than two identifier variables are supported.
```

Blimp has interpreted more than two identifier variables in the MODEL command. Blimp currently only allows a maximum of three-levels (i.e., two identifier variables).

```
ERROR: Variable X1 has only one observed category.
       Ordinal and nominal variables need
       a minimum of two observed categories.
```

The variable 'X1' (where 'X1' could be any variable name) was listed on the ORDINAL or NOMINAL command, but has only one observed category. Blimp requires more than one category to be observed in order to impute the variable.

```
ERROR: Variable: X1 not in data.
```

The variable 'X1' (where 'X1' could be any variable name) was listed in the MODEL command and not in the VARIABLES command.

```
ERROR: Identifier variable: SUBJID not in data.
```

The identifier variable SUBJID (where SUBJID could be any identifier variable name) listed prior to the tilde in the MODEL command does not appear in the VARIABLES command.

```
ERROR: Identifier variables have the same number of clusters.
       Cross-classified models are currently unsupported.
```

Blimp has detected the identifier variables have the same number of clusters. At this time, Blimp does not support cross-classified models.

```
ERROR: Identifier's appear to be a cross-classified model.
       Cross-classified models are currently unsupported.
```

Blimp has detected the identifier variables do not follow the proper nesting structure. At this time, Blimp does not support cross-classified models.

```
ERROR: Please place ONE asterisk (*) in DATA command.
```

More than one asterisk (*) was placed in the file path given in the DATA command. This can also be triggered when no asterisk is found. An asterisk is required for the SIMULATE command.

```
ERROR: Failed to read the data in,
       please use a comma separated file
       or space separated file.
```

Blimp was unable to read the data. This could be due to the file path specified not being correct or due to the file type not being recognized.

```
ERROR: The number of variables listed does not equal data columns.
```

The number of variables listed in the VARIABLES command does not equal the number of columns the data matrix read in by Blimp.

```
ERROR: No missing variables in MODEL command.
```

There were no missing variables listed in the MODEL command.

```
ERROR: A matrix is numerically positive indefinite
       Either:
              (1) Try another seed.
              (2) Specify fewer random effects.
              (3) Specify fewer variables in model.
```

During the imputation process, a matrix became numerically positive indefinite and Blimp was unable to continue because of an inversion problem. It is recommended to first try another seed. If this continues, then try to specify fewer random effects. Finally, try specifying fewer variables.

```
   Chain i failed with status 2.
```

Chain `i`, where `i` is a number, had a matrix become computationally positive indefinite. If this is the case, it is recommended to first try another seed. If this continues, then try to specify fewer random effects. Finally, try specifying fewer variables.

```
   Replication i return an error code of 1.
```

During the simulation, the replication `i`, where `i` is a number, had an error and returned the code of error code of 1. An error code of 1 indicates a failure to load the data set.

```
   Replication i return an error code of 2.
```

During the simulation, the replication `i`, where `i` is a number, had an error and returned the code of error code of 2. An error code of 2 indicates that a matrix became numerically positive indefinite. If this is the case, it is recommended to first try another seed.

```
ERROR: A non-numeric value of '.' was given in MISSING command.
       Currently only numeric missing codes are allowed.
```

Blimp only accepts numerical values as missing data codes at this time.

Warning Messages

```
WARNING: Overriding OUTFILE command in input.
```

An argument from the command-line is overriding the file path given in the `OUTFILE` command (e.g., when implementing an external simulation). If this is not desired, see Chapter 4.

```
WARNING: Overriding SEED command in input.
```

An argument from the command-line is overriding the value given in the `SEED` command (e.g., when implementing an external simulation). If this is not desired, see Chapter 4.

```
WARNING: Overriding DATA command in input.
```

An argument from the command-line is overriding the file path given in the `DATA` command (e.g., when implementing an external simulation). If this is not desired, see Chapter 4.

```
WARNING: Entering simulation mode.
```

Simulation mode was enabled with a `SIMULATE` command.

```
WARNING: Separate data files are not available for simulation mode.
```

The `separate` keyword was specified with the `SIMULATE` command. The `SIMULATE` command will override the `separate` keyword.

```
WARNING: Zero or less imputations requested. Defaulting to one imputation.
```

The number of imputations requested were read in to be zero or a negative number. Blimp is setting the number of imputations to one.

```
WARNING: Less than zero thinning interval requested. Defaulting to zero.
```

The thinning interval must be greater than or equal to zero. Blimp is defaulting to zero.

```
WARNING: Less than zero burn-in requested. Defaulting to zero.
```

The burn-in interval must be greater than or equal to zero. Blimp is defaulting to zero.

```
WARNING: Maximum number of processors allowed is 4.
         This is based on your hardware specifications.
         Setting the processors to 4.
```

Blimp will not allow more processors requested than number of physical CPU cores in the computer (note in the above warning 4 may change depending on the hardware). Blimp will default to the maximum number of allowed processors.

```
WARNING: Multithreading not enabled.
         Setting the processors to 1.
```

The distribution of Blimp being used does not have multithreading enabled. Therefore, Blimp is setting itself to one processor.

```
WARNING: A minimum of 100 burn-in needed for PSR.
         Setting burn-in to 100.
```

The `psr` keyword was specified in the `OPTIONS` command and the burn-in requested was less than 100. Blimp prints the PSR statistic for every 100 burn-in iterations. Therefore, Blimp is defaulting to a minimum of 100 burn-in iterations.

```
NOTE: Setting number of imputations to 2
      to match number of chains requested.
```

More chains were requested than imputations. A minimum of one imputation per chain must be requested. Blimp is defaulting the number of imputations to the number of chains requested.

```
WARNING: Multithreading not enabled.
         Reverting to single thread simulation.
```

The distribution of Blimp being used does not have multithreading enabled. Therefore, Blimp is using a single-threaded simulation.

```
WARNING: Variable "X1" in ORDINAL command was not used.
         Cannot be found in VARIABLE command.
```

The variable 'X1' (note this will be the name of variable causing the warning message) was used in the `ORDINAL` command, but is not listed in the `VARIABLE` command. It will be ignored.

```
WARNING: Ignoring variable "X1" in ORDINAL command is not in MODEL command.
         Cannot be found in MODEL statement.
```

The variable 'X1' (note this will be the name of variable causing the warning message) was used in the `ORDINAL` command, but is not listed in the `MODEL` command. It will be left out of the model and ignored.

```
WARNING: Variable "X1" in NOMINAL command was not used.
         Cannot be found in VARIABLE command.
```

The variable 'X1' (note this will be the name of variable causing the warning message) was used in the `NOMINAL` command, but is not listed in the `VARIABLE` command. It will be ignored.

```
WARNING: Ignoring variable "X1" in NOMINAL command is not in MODEL command.
         Cannot be found in MODEL statement.
```

The variable 'X1' (note this will be the name of variable causing the warning message) was used in the `NOMINAL` command, but is not listed in the `MODEL` command. It will be left out of the model and ignored.

```
WARNING: No identifier variable specified.
         Defaulting to single-level imputation.
```

Blimp did not interpret an identifier variable in the `MODEL` command. Blimp will treat the file as a single-level imputation. If this is desired, ignore this warning. Otherwise, check the syntax's `MODEL` command. See Chapter 3 on specifying a `MODEL` command.

```
WARNING: 5 observations have all variables in the imputation
         model missing. They have been dropped from data set.
```

Blimp has dropped 5 observations, where 5 is the number specific to the data set. Blimp will not impute variables with missing observations on all variables in the `MODEL` command.

```
WARNING: Excluding the following variables as predictors at the listed level.
           Variable "X1" excluded from level 2 imputation.

         These variables are either orthogonal to all variables at that level
         (e.g., because they lack variation at that level) or their cluster
         mean is linearly dependent with another variable at that level.
```

The variable 'X1' (note this will be the name of variable causing the warning message) is being excluded from the imputation model at the level specified in the message (i.e., 2 in the above example message).

References

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.

Enders, C., Keller, B., & Levy, R. (2017). A fully conditional specification approach to multilevel imputation of categorical and continuous variables. *Psychological methods*.

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical science*, 457–472.

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*, *23*(2), 93–116.

Montague, M., Krawec, J., Enders, C., & Dietz, S. (2014). The effects of cognitive strategy instruction on math problem solving of middle-school students of varying ability. *Journal of Educational Psychology*, *106*(2), 469.

van Buuren, S., et al. (2011). Multiple imputation of multilevel data. *Handbook of advanced multilevel analysis*, 173–196.

Zhang, Q., & Wang, L. (2016). Moderation analysis with missing data in the predictors. *Psychological Methods, Advanced online publication*.