$\mathfrak{A}melia$ II: A Program for Missing Data

James Honaker, Gary King, and Matthew Blackwell

January 15, 2007

Contents

1 Introduction **3**

2 What 𝔄*melia* Does **4**

3 Versions of 𝔄*melia* **5**

4 Installation and Updates **5**
4.1 Windows — AmeliaView 5
4.2 Windows — 𝔄*melia II* for R 5
4.3 Linux 6

5 Program Overview **6**
5.1 AmeliaView 6
5.2 𝔄*melia* for R 7

6 Data Input and Output **7**
6.1 AmeliaView 7
6.2 𝔄*melia* for R 8

7 Options **9**
7.1 Screen Output 9
7.2 Transformations of Variables 9
7.2.1 Ordinal 10
7.2.2 Nominal 10
7.2.3 Natural Log 11
7.2.4 Square Root 11
7.2.5 Logistic 11
7.3 Identification Variables 11
7.4 Time Series, or Time Series Cross Sectional Data 12

8 Setting Priors **13**
8.1 Empirical (Ridge) Priors for High Missingness, Small n's, or Large Correlations 13
8.2 Observation-level priors 13

9 Diagnostics **14**
9.1 Compare 14
9.2 Overimpute 15
9.3 Overdispersed Starting Values 18

10 Sessions **19**

11 Error Checking and Reference Guide **22**
11.1 Error Code Reference 22

12 Command Line Argument Reference **24**
12.1 `amelia`: Multiple Imputation of Incomplete Data 24

13 AmeliaView Menu Guide **28**
13.1 Step 1 - Input 28
13.2 Step 2 - Options 28
13.2.1 Variables Dialog 29
13.2.2 Time Series Cross Sectional Dialog 30
13.2.3 Priors Dialog 31
13.2.4 Case Priors 31
13.2.5 Observational Priors 32
13.2.6 Add Distribution Prior 33
13.2.7 Add Range Prior 33
13.3 Step 3 - Output 34
13.3.1 Diagnostics Dialog 35

1 Introduction

This manual presents a brief introduction to the $\mathfrak{A}melia\ II$ software package for multiple imputation. $\mathfrak{A}melia\ II$ allows users to appropriately impute ("fill in" or rectangularize) incomplete data sets so that analyses which require complete observations can use all the information present in a dataset with missingness, and avoid the biases and inefficiencies that can result from dropping all partially observed observations from the analysis.

James Honaker, Anne Joseph, Gary King, Kenneth Scheve, and Naunihal Singh (1998) wrote the first version of Amelia to remedy the discrepancy between the way social scientists then analyzed data with missing values and the recommendations of the statistics community. With a few notable exceptions, statisticians and methodologists had agreed on the advantages of the concept of "multiple imputation" as a general-purpose approach to data with missing values; yet at the time most social scientists had still used listwise deletion (deleting all observations with at least one missing cell) or other ad hoc techniques like best guess or mean imputation to make inferences in the presence of missing data. These practices are known to be inefficient and often biased. Unfortunately, many of the multiple imputation algorithms that had been proposed in the literature were unfit for handling the common problems of social science data and were often difficult to use, even for experts.

A key reason multiple imputation had not been used frequently in the social sciences is because it was developed so that producers of large public-use data sets could conduct the imputation for the analyst, but this paradigm did not help those who collected, merged, or processed their own data. Before King, Honaker, Joseph and Scheve (2001), a small number of public use data sets had been imputed, but very few researchers had imputed a data set or used a multiply imputed data set. The contribution of the work that led to the first version of $\mathfrak{A}melia$ was the EMis algorithm that produced imputations requiring less expertise to use and with more speed than existing MCMC-based algorithms. This made it possible to write $\mathfrak{A}melia$ so that scholars could impute their own data, rather than waiting for others to do it for them.

The $\mathfrak{A}melia\ II$ program goes several significant steps beyond the capabilities of the first version of Amelia. For one, the bootstrapping-based EMB algorithm included in $\mathfrak{A}melia\ II$ can impute many more variables, with many more observations, in much less time. The great simplicity and power of the EMB algorithm made it possible to write $\mathfrak{A}melia\ II$ so that it virtually never crashes — which to our knowledge makes it unique among all existing multiple imputation software.

$\mathfrak{A}melia\ II$ also has features to make valid and much more accurate imputations for cross-sectional, time-series, and time-series-cross-section data, and allows the incorporation of observation and data-matrix-cell level prior information. This software implements the ideas developed in **?**. Please read this paper in conjunction with this manual.

2 What $\mathfrak{A}melia$ Does

Multiple imputation involves imputing m values for each missing cell in your data matrix and creating m "completed" data sets. (Across these completed data sets, the observed values are the same, but the missing values are filled in with different imputations that reflect the uncertainty about the missing data.) After imputation with $\mathfrak{A}melia\ II$'s EMB algorithm, you can apply whatever statistical method you would have used if there had been no missing values to each of the m data sets, and use a simple procedure, described in the next paragraph, to combine the results. (You can combine the results automatically by doing your data analyses within Zelig, or within Clarify for Stata; see `http://gking.harvard.edu/stats.shtml`.) Under normal circumstances, you only need to impute once and can then analyze the m imputed data sets as many times and for as many purposes as you wish. The advantage of $\mathfrak{A}melia\ II$ is that it combines the comparative speed and ease-of-use of our algorithm with the power of multiple imputation, to let you focus on your substantive research questions rather than spending time developing complex application-specific models for nonresponse in each new data set. Unless the rate of missingness is very high, $m = 5$ (the program default) is probably adequate.

In order to combine the results across m data sets, first decide on the quantity of interest to compute, such as univariate mean, regression coefficient, predicted probability, or first difference. Then, the easiest way is to draw $1/m$ simulations of q from each of the m data sets, combine them into one set of m simulations, and then to use the standard simulation-based methods of interpretation common for single data sets (King, Tomz and Wittenberg, 2000).

Alternatively, you can combine directly and use as the multiple imputation estimate of this parameter, $\bar{q}$, the average of the m separate estimates, q_j $(j = 1, ..., m)$:

$$\bar{q} = \frac{1}{m} \sum_{j=1}^{m} q_j. \tag{1}$$

The variance of the point estimate is the average of the estimated variances from *within* each completed data set, plus the sample variance in the point estimates *across* the data sets (multiplied by a factor that corrects for the bias because $m < \infty$). Let $SE(q_j)^2$ denote the estimated variance (squared standard error) of q_j from the data set j, and $S_q^2 = \Sigma_{j=1}^{m}(q_j - \bar{q})^2/(m-1)$ be the sample variance across the m point estimates. The standard error of the multiple imputation point estimate is the square root of

$$SE(q)^2 = \frac{1}{m} \sum_{j=1}^{m} SE(q_j)^2 + S_q^2(1 + 1/m). \tag{2}$$

Users should see especially Pp. 57-58 of King et al. (2001) for a variety of practical suggestions in making imputations, such as what variables to include in the imputation stage, how to keep imputations within logically possible ranges, etc.

3 Versions of $\mathfrak{A}melia$

Two versions of $\mathfrak{A}melia\ II$ are available, each with its own advantages and drawbacks. First, $\mathfrak{A}melia\ II$ exists as a package, or collection of functions, for the R statistical software package. Users can utilize their knowledge of the R language to run $\mathfrak{A}melia\ II$ at the command line or to create scripts that will run $\mathfrak{A}melia\ II$ and preserve the commands for future use. Alternatively, AmeliaView, an interactive Graphical User Interface (GUI), allows users to set options and run $\mathfrak{A}melia$ without any knowledge of the R programming language. AmeliaView enables users to set all of the $\mathfrak{A}melia\ II$ options available and enables those with limited or no coding experience to create expert level imputations without knowledge of the R language.

Both versions of $\mathfrak{A}melia\ II$ are available on the Windows and Linux platforms and $\mathfrak{A}melia\ II$ for R runs in any environment that R can. All versions of $\mathfrak{A}melia$ require the R software, which is freely available at `http://www.r-project.org/`.

4 Installation and Updates

Before installing $\mathfrak{A}melia\ II$, you must have installed R version 2.1.0 or higher, which is freely available at `http://www.r-project.org/`.

4.1 Windows — AmeliaView

To install AmeliaView in the Windows environment, simply download the installer `setup.exe` from `http://gking.harvard.edu/amelia/` and run it. The installer will ask you to choose a location to install $\mathfrak{A}melia\ II$. If you have installed R with the default options, $\mathfrak{A}melia\ II$ will automatically find the location of R. If the installer cannot find R, it will ask you to locate the directory of the most current version of R. Make sure you choose the directory name that includes the version number of R (e.g. C:/Program Files/R/R-2.4.0) and contains a subdirectory named `bin`. The installer will also put shortcuts on your Desktop and Start Menu.

4.2 Windows — $\mathfrak{A}melia\ II$ for R

Users familiar with the R language, and who intend to use $\mathfrak{A}melia$ primarily as a function within other R code can install all the Amelia functions as one would install any other R library. A package install can be called within R by:

```
> install.packages("Amelia",repos="http://gking.harvard.edu")
```

While Amelia works in builds of R beyond version 2.1.0, the online package install will only be available for the most recent version of R, currently, 2.4.0.

Even users familiar with the R language may find it useful to utilize AmeliaView to set options on variables, change arguments, or run diagnostics. From the command line, AmeliaView can be brought up with the call:

```
> AmeliaView()
```

4.3 Linux

To install *Amelia* in a Linux OS, you must install the *Amelia* library into your version of R. This is true even if users only wish to use AmeliaView. Download the package from `http://gking.harvard.edu/amelia`. Then, in the same directory as the file, at the Linux/Unix command line type

```
> R CMD INSTALL --library=.R/library amelia_1.0.tar.gz .
```

If you do not have access to the root, you can install the package locally. Create a directory (i.e. myrlibrary) to be the local storage space for *R* packages. Once this directory is created you can install the package to that local library:

```
> R CMD INSTALL --library=~/myrlibrary amelia_1.0.tar.gz .
```

Once this is complete you need to edit or create your *R* profile. Locate or create `/.Rprofile` in your home directory and add this line:

```
.libPath(``~/myrlibrary'')
```

This will add your local library to the list of library paths that R searches in when you load libraries.

Linux users can use AmeliaView in the same way as Windows users of *Amelia* for R. From the command line, AmeliaView can be brought up with the call:

```
> AmeliaView()
```

5 Program Overview

5.1 AmeliaView

Built around the core of the *R* library, AmeliaView has a graphical interface for users to input data, set options, create the imputed data sets, and run diagnostics. The structure of the program moves logically through these steps. In Windows, this can be accessed from the desktop shortcut created by the installer. In Linux, accessing the GUI requires calling a function from within R[1]. Once AmeliaView is open, no direct use of *R* is required.

The main window of AmeliaView is divided into three steps. In Step 1, you indicate the location of your data and load it into the program. In Step 2, you can specify options about your data by using the Options dialogs. In Step 3, you can configure the output of *Amelia* and execute the *Amelia* code. You have the option to save your output data files for further use.

In most simple applications, all you will need to do is load an input data file, set any options you desire, and hit the "Run Amelia" button. For example:

[1] AmeliaView can be called from within *R* in Windows, as well, using the `AmeliaView()` function.

1. Specify the type of data file you wish to input by using the "Input Data Type" drop-down menu. Next, use the "Browse..." button to find the location of your data file. Once this is specified, you can hit the "Load Data" button to load the data. If your data loads correctly, the status bar at the bottom of the program will show the filename, number of rows and number of columns. At this point you can use the "Summarize Data" button to view summary statistics about the variables along with histogram plots of them.

2. In the Options step, you can specify the time series and cross sectional variables in your data set (if any) by using the appropriate drop-down menus. Each of the "Variables", "TSCS", and "Priors" buttons open a separate dialog box in which you can set options in each of these categories.

3. In the Output step, you can specify what file type (if any) in which you would like to save your output data. You can also set the name of the output data and the number of data sets you would like. You can now run *Amelia* by pressing the "Run Amelia" button. A dialog will open that tracks the progress of *Amelia* and will let you know when it has finished. Once this is complete, you can either use the diagnostics or close *Amelia*.

5.2 *Amelia* for R

Using *Amelia* in *R* you will first have to load the library into R using the `library(Amelia)` command. Once the package is loaded, you can set options using the arguments of the `amelia()` function. The output data will be either a list of m data frames or matrices depending on your input data. Please refer to the end of this manual for detailed documentation on adjusting the optional arguments from their default values.

6 Data Input and Output

6.1 AmeliaView

AmeliaView uses the `foreign` package in *R* to import and export various types of files. In the Input box, you can choose the file type you wish to use from the group of supported file types: Comma Separated Values (.csv), Tab-Delimited (.txt), Stata (.dta), SPSS (.dat), and SAS Transport (.xport). Once this is set, you can proceed to locate your file in one of two ways. You can type the location of the file in the "Input Data File" entry and press the "Load Data" button. Or you can locate your file by using the "Browse..." button to select the file of your choice. Once you have entered the file name, press the "Load Data" button to load the file into *Amelia*.

When you have loaded the data, you can use the "Summarize Data" button to see the data. This includes seeing the minimum and maximum values along with the mean and standard deviation. Another feature is the ability to view a plot of

the histogram of each individual variable. This can help you get a graphical sense of the observed values in your dataset.

Fewer options exist for the output data files due to the limitations of the `foreign` package. You can only save the imputed datasets as either a CSV, Tab-Delimited, or Stata file. However, most statistical packages will allow you to read in datasets as either CSV or Tab-delimited. The names of the files will be the name you specify, plus a number appended at the end to distinguish between successive imputed datasets. For example, if you set the name to be "mydata" and the output file type to be CSV, your files would be:

```
mydata1.csv
mydata2.csv
...
```

There will be as many datasets as you indicate in the output options box.

Another output file that is produced when running *Amelia* is the Replication Archive. This file can be used to reproduce the same set of options you ran *Amelia* with so that you or anyone else can replicate the same findings. This file automatically saves to `amarchive.r`. (If, as is becoming standard practice, you create and archive replication data sets to accompany your papers (King, 1995), you should include this file.) This file can also serve as a session save that can be opened later to load the same options that you ran *Amelia* under. This allows you to pick up where you left off in the next run of *Amelia*, adjust options and rerun the imputations, or return at a later point and use the diagnostic routines.

6.2 *Amelia* for R

Data input in the command-line version of *Amelia* for R is identical to any data input in R. You must have your data in either a text format such as Comma Separated Values (.csv) or Tab-Delimited (.txt) and then read them into *R* (generally this would involve the functions `read.csv` or `read.table`, respectively). You can read about the specifics of how to load data into *R* in the *R* documentation or in Zelig (`http://gking.harvard.edu/zelig`). Of course your data may be generated by code and amelia called as a function later in that code.

Besides various commercial packages to transfer your data between formats, a viable option is using the `foreign` package. This package can greatly expand the number of different formats that *R* can input, including Stata, SPSS, and SAS transport.

Whichever way you get your data into *R*, when passing it to the `amelia` function, it can either be in the form of a data frame or matrix. After *Amelia* has run, you will get datasets returned in the same type as the format of the dataset given to the amelia function. Once you have these datasets, you can manipulate them in *R* or save them to files using the *R* function `write`.

For example, a simple session or code fragment might be:

```
> library(amelia)
> x<-read.table("mydata.csv")
```

```
> output<-amelia(data=x,empri=10,ords=c(3,4))
> save(output,file="output.rData")
```

where `ords` and `empri` are options detailed later in this manual.

7 Options

There are a variety of options in *Amelia* that customize the imputation model to handle problems that are common in social science data. In AmeliaView, these options are set using the dialog boxes "Variables," "TSCS," and "Priors." At these dialogs, the user can visually inspect and set each option. Please refer to the Menu Guide below for details on the content of the dialogs.

In *Amelia* for R, these options must be set on the command line. From within R, users can set the transformations by including a vector of column numbers or names (if column names exist), that *Amelia* should transform. For example,

```
> amelia(mydata, noms=c(2,3,7))
```

or

```
> amelia(mydata, logs=c(``gdp'',``population''))
```

7.1 Screen Output

The output of the imputation model in the AmeliaView program appears in a new window. It lists stages of the imputation model, including the number of iterations of the EM chain necessary in each of the bootstraps.

Screen output in the *R* command level can be adjusted with the "print to screen" argument, `p2s`. At a value of 0, no screen printing will occur. This may be useful in large jobs or simulations where a very large number of imputation models may be required. The default value of 1, lists each bootstrap, and displays the number of iterations required to reach convergence in that bootstrapped dataset. The value of 2 gives more thorough screen output, including, at each iteration, the number of parameters that have significantly changed since the last iteration. This may beuseful when the EM chain length is very long, as it can provide an intuition for many parameters still need to converge in the EM chain, and a sense of the time remaining. However, it is worth noting that the last several parameters can often take a significant fraction of the total number of iterations to converge. Setting `p2s` to 2 will also generate information on how EM algorithm is behaving, such as a `'!'` when the current estimated complete data covariance matrix is not invertible and a `'*'` when the likelihood has not monotonically increased in that step.

7.2 Transformations of Variables

Social science data commonly includes variables that fail to fit to a multivariate normal distribution. Indeed, numerous models have been introduced specifically to

deal with the problems they present. As it turns out, much evidence in the literature (discussed in King et al. 2001) indicates that the multivariate normal model used in $\mathfrak{A}melia$ usually works well for the imputation stage even when discrete or non-normal variables are included and when the analysis stage involves these limited dependent variable models. Nevertheless, $\mathfrak{A}melia$ includes some limited capacity to deal directly with ordinal and nominal variables and to variables that require other transformations. In general nominal and log transform variables must be declared to $\mathfrak{A}melia$, whereas ordinal (including dichotomous) variables often need not be, as described below. (For harder cases, see (Schafer, 1997), for specialized MCMC-based imputation models for discrete variables.)

Although these transformations are taken internally on these variables to better fit the data to the multivariate normal assumptions of the imputation model, all the imputations that are created will be returned in the original untransformed form of the data (If the user has *already* performed transformations on their data (such as taking a log or square root) these do not need to be declared, as that would result in the transformation occurring *doubly* in the imputation model). The fully imputed datasets that are returned will always be in the form of the original data that is passed to the amelia routine.

7.2.1 Ordinal

In much statistical research, researchers treat independent ordinal (including dichotomous) variables as if they were really continuous. If the analysis model to be employed is of this type, then nothing extra is required of the of the imputation model. Users are advised to allow $\mathfrak{A}melia$ to impute non-integer values for any missing data, and to use these non-integer values in their analysis. Sometimes this makes sense, and sometimes this defies intuition. One particular imputation of 2.35 for a missing value on a seven point scale carries the intuition that the respondent is between a 2 and a 3 and most probably would have responded 2 had the data been observed. This is easier to accept than an imputation of 0.79 for a dichotomous variable where a zero represents a male and a one represents a female respondent. However, in both cases the non-integer imputations carry more information about the underlying distribution than would be carried if we were to force the imputations to be integers. Thus whenever the analysis model permits, missing ordinal observations should be allowed to take on continuously valued imputations.

Often, however, analysis models require some variables to be strictly ordinal, as for example the dependent variable must be in a logistical regression. Imputations for variables set as ordinal are created by taking the continuously valued imputation and using an appropriately scaled version of this as the probability of success in a binomial distribution. The draw from this binomial distribution is then translated back into one of the ordinal categories.

7.2.2 Nominal

Nominal variables (other than dichotomous) must be treated quite differently than ordinal variables. Any multinomial variables in the data set (such as religion coded 1 for Catholic, 2 for Jewish, and 3 for Protestant) must be specified to $\mathfrak{A}melia$.

For a p-category multinomial variable, $\mathfrak{A}melia$ will determine p (as long as your data contain at least one value in each category), and substitute $p-1$ binary variables to specify each possible category. These new $p-1$ variables will be treated as the other variables in the multivariate normal imputation method chosen, and receive continuous imputations. These continuously valued imputations will then be appropriately scaled into probabilities for each of the p possible categories, and one of these categories will be drawn, where upon the original p-category multinomial variable will be reconstructed and returned to the user. Thus all imputations will be appropriately multinomial.

Since $\mathfrak{A}melia$ properly treats a p-category multinomial variable as $p-1$ variables, one should understand the number of parameters that are quickly accumulating if many multinomial variables are being used. If the square of the number of real and constructed variables is large relative to the number of observations, the user is recommended to implement a ridge prior distribution on the parameter space.

7.2.3 Natural Log

If one of your variables is heavily skewed or has outliers that may alter the imputation in an unwanted way, you can use a natural logarithm transformation of that variable in order to normalize its distribution. This transformed distribution helps $\mathfrak{A}melia$ to avoid imputing values that depend too heavily on outlying data points. Log transformations are common in expenditure and economic variables where we have strong beliefs that the marginal relationship between two variables decreases as we move across the range.

7.2.4 Square Root

Event count data is often heavily skewed and has nonlinear relationships with other variables. One common transformation to tailor the linear model to count data is to take the square roots of the counts. This is a transformation that can be set as an option in amelia.

7.2.5 Logistic

Proportional data is sharply bounded between 0 and 1. A logistic transformation is one possible option in amelia to make the distribution symmetric and relatively unbounded.

7.3 Identification Variables

Datasets often contain identification variables, such as country names, respondent numbers, or other id numbers, codes or abbreviations. Sometimes these are text and sometimes these are numeric. Often it is not appropriate to include these variables in the imputation model, but it is useful to have them remain in the imputed datasets (However, there are models that would include the ID variables in the imputation model, such as fixed effects model for data with repeated observations of the same countries). Identification variables which are not to be included in the imputation

model can be identified with the argument `idvars`. These variables will not be used in the imputation model, but will be kept in the imputed datasets.

In order to conserve memory, it is wise to remove unnecessary variables from a data set before loading it into *Amelia*. The only variables you should include in your data when running *Amelia* are variables you will use in the analysis stage and those variables that will help in the imputation model. While it may be tempting to simply mark unneeded variables as IDs, it only serves to waste memory and slow down the imputation procedure.

7.4 Time Series, or Time Series Cross Sectional Data

Many variables that are recorded over time within a cross-sectional unit are observed to vary smoothly over time. In such cases, knowing the observed values of observations close in time to any missing value may enormously aid the imputation of that value. However, the exact pattern may vary over time within any cross-section. There may be periods of growth, stability, or decline; in each of which the observed values would be used in a different fashion to impute missing values. Also, these patterns may vary enormously across different cross-sections, or may exist in some and not others. *Amelia* can build a general model of patterns within variables across time by creating a sequence of polynomials of the time index. If, for example, *GDP* varies smoothly over time, then we make the modeling assumption that there exists some polynomial that describes the economy in cross-sectional unit i at time t as:

$$\text{GDP}(t)_i = \beta_0 + \beta_1 t + \beta_1 t^2 + \beta_1 t^3 \ldots \tag{3}$$

And thus if we include enough higher order terms of time then the pattern between observed values of GDP can be estimated. *Amelia* will create polynomials of time up to the user defined k-th order, ($k \leq 3$). If cross-sectional units are specified these polynomials can be interacted with the cross-section unit (this is the default) to allow the patterns over time to vary between cross-sectional units. Unless you strongly believe all units have the same patterns over time in all variables (including the same constant term), this is a reasonable default. When k is set to 0, this interaction simply results in a model of *fixed effects* where every unit has a uniquely estimated constant term. *Amelia II* does not smooth the observed data, and only uses this functional form, or one you choose, with all the other variables in the analysis and the uncertainty of the prediction, to impute the missing values.

In AmeliaView, the TSCS settings can be adjusted in the Time Series Cross-Sectional Dialog (discussed in section 13.2.2 of this manual). At the *R* command line, these options can be set as:

```
> amelia(mydata, ts=1, cs=2, polytime=2, intercs=TRUE)
```

Where `ts` and `cs` give the locations of the time and cross-section indicators (often country names and years, respectively), `polytime` is an integer between 0 and 3 inclusive, that sets the order of polynomials to use, and `intercs` defines whether these polynomials should be interacted with the cross-sectional unit.

8 Setting Priors

Amelia has a number of methods of setting priors within the imputation model. Two of these are commonly used and discussed below, empirical or ridge priors and observational priors.

8.1 Empirical (Ridge) Priors for High Missingness, Small n's, or Large Correlations

When the data to be analyzed contain a high degree of missingness or very strong correlations among the variables, or when the number of observations is only slightly greater than the number of parameters $p(p+3)/2$ (where p is the number of variables), results from your analysis model will be more dependent on the choice of imputation model. This suggests more testing in these cases of alternative specifications under *Amelia*.

In these circumstances, we recommend adding a ridge prior which will help with numerical stability by shrinking the covariances among the variables toward zero without changing the means or variances. The ridge prior can be implemented by setting the option `empri` in R and the Ridge Prior entry under the "Priors" button in the standalone. Including this prior as a positive number is roughly equivalent to adding `empri` artificial observations to the data set with the same means and variances as the existing data but with zero covariances. Thus, increasing the `empri` setting results in more shrinkage of the covariances, thus putting more a priori structure on the estimation problem: like many Bayesian methods, it reduces variance in return for an increase in bias that one hopes does not overwhelm the advantages in efficiency. In general, we suggest keeping the value on this prior relatively small and increase it only when necessary. A recommendation of 0.5 to 1 percent of the number of observations, n, is a reasonable starting value, and often useful in large datasets to add some numerical stability. For example, in a dataset of two thousand observations, this would translate to a prior value of 10 or 20 respectively. A prior of up to 5 percent is moderate in most applications.

8.2 Observation-level priors

Researchers often have additional prior information about missing data values based on previous research, academic consensus, or personal experience. Amelia can incorporate this information to produce vastly improved imputations. The Amelia algorithm allows users to include informative Bayesian priors about individual missing data cells instead of the more general model parameters, many of which have little direct meaning.

The incorporation of priors follows basic Bayesian analysis where the imputation turns out to be a weighted average of the model-based imputation and the prior mean, where the weights are functions of the relative strength of the data and prior: when the model predicts very well, the imputation will down-weight the prior, and vice versa (**?**).

The priors about individual observations should describe the analyst's belief about the distribution of the missing data cell. This can either take the form of a mean and a standard deviation or a confidence interval. For instance, we might know that GDP growth for Ghana in a given year was somewhere around 3.5%, but we have some uncertainty as to the exact value. Our prior belief about the distribution of the missing data cell, then, centers on 3.5 with a standard deviation that reflects the amount of uncertainty we have about our prior belief.

In AmeliaView, the observational priors dialog will guide you through adding priors to the imputation. Refer to section 13.2.5 for further details on how to add priors in AmeliaView.

Alternatively, in R, you must build a priors matrix with either four or five columns. Each row of the matrix represents a prior on either one observation or one variable. In any row, the entry in the first column is the row of the observation and the entry is the second column is the column of the observation. In the four column priors matrix the third and fourth columns are the mean and standard deviation of the prior distribution of the missing value. For example, in the following example the first row constructs a prior mean of 500 with a standard deviation of 50 on the missing cell in row 42, column 3:

```
> priors
     row column mean std dev
[1,]  42      3  500      50
[2,]  46      2    1       3
[3,]   0      2    0       1
```

In the five column matrix, the last three columns describe a confidence range of the data. The columns are a lower bound, an upper bound, and a confidence level between 0 and 1, exclusive. Whichever format you choose, it must be consistent across the entire matrix. The last prior created in the example above specifies a row number of 0. This indicates that the prior created should be applied to all missing values in this variable (here variable 2), unless another prior is specifically created for that observation (as in observation 46 in this example).

9 Diagnostics

𝔄*melia* currently provides three diagnostic tools to inspect the imputations that are created. These routines `compare`, `overimpute` and `overdisperse` graphically investigate the distribution of the imputations, the fit of the imputation model, and modality of the Likelihood space optimized by the EM chain, respectively.

9.1 Compare

In the diagnostic window of AmeliaView, the compare function will, for a given variable, generate a plot of the relative frequencies of the observed data with an overlay of the relative frequency of the imputed values. The imputed curve plots the density of the *mean* imputation over the m datasets. That is, for each cell that

is missing in the variable the diagnostic will find the mean of that cell in each of the m datasets and use that value for the density plot. These graphs will allow you to inspect how the density of imputations compares to the density of observed data. Some discussion of these graphs can be found in Abayomi, Gelman and Levy (2005). Minimally, these graphs can be used to check that the mean imputation falls within known bounds, when such bounds exist in certain variables or settings.

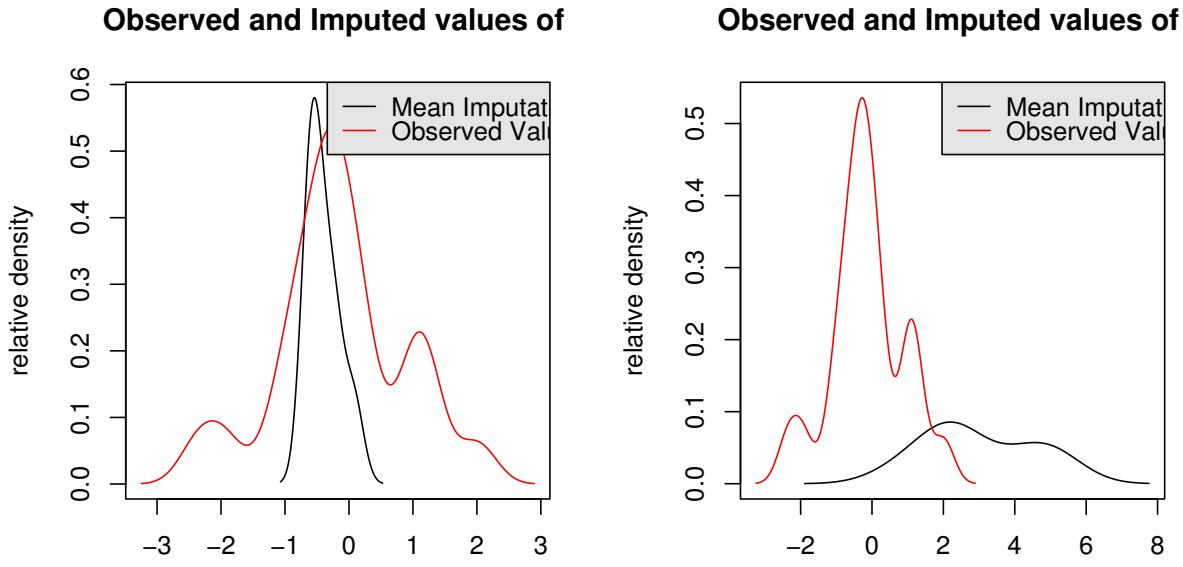

Figure 1: Two examples of the compare diagnostic graph. In the left graph all the imputations, the density of which is shown in black, are contained within the range of the observed data for this variable (in red). In the right graph some of the mean imputed values are larger than the largest observed value. This may be reasonable (if for example, high income respondents to a survey do not report their income) or may be a warning to change the imputation model, if for example the data has strong bounds and these imputations fall well outside these bounds.

This graph can also be called from *R* as:

```
> output<-amelia(data=x)
> compare.density(data=x,output=output,var=3)
```

where `var` is the column number (or the name) of the variable you would like to produce the graph for.

9.2 Overimpute

Overimputing is a technique we have developed to judge the fit of the imputation model. Because of the nature of the missing data mechanism, it is impossible to tell whether the mean prediction of the imputation model is close to the unobserved

value that is trying to be recovered. By definition this missing data does not exist to create this comparison, and if it existed we would no longer need the imputations or care about their accuracy. However, a natural question the applied researcher will often ask is how accurate are these imputed values?

Overimputing involves sequentially treating each of the *observed* values as if they had actually been missing. For each observed value in turn we then generate several hundred imputed values of that observed value, *as if it had been missing.* While $m = 5$ imputations are sufficient for most analysis models, this large number of imputations allows us to construct a confidence interval of what the imputed value would have been, had any of the observed data been missing. We can then graphically inspect whether our observed data tends to fall within the region where it would have been imputed had it been missing.

Our overimputation diagnostic runs this procedure through all of the observed values for a user selected variable. We can graph the estimates of each observation against the true values of the observation. On this graph, a $y = x$ line indicates the line of perfect agreement; that is, if the imputation model was a perfect predictor of the true value, all the imputations would fall on this line. For each observation, $\mathfrak{A}melia$ also plots 90% confidence intervals that allows the user to visually inspect the behavior of the imputation model. By checking how many of the confidence intervals cover the $y = x$ line, we can tell how often the imputation model can confidently predict the true value of the observation.

Occasionally, the overimputation can display unintuitive results. For example, different observations may have different numbers of observed covariates. If covariates that are useful to the prediction are themselves missing, then the confidence interval for this observation will be much larger. In the extreme, there may be observations where the observed value we are trying to overimpute is *the only* observed value in that observation, and thus there is nothing left to impute that observation with when we pretend that it is missing, other than the mean and variance of that variable. In these cases, we should correctly expect the confidence interval to be very large.

An example of this graph is shown in figure 2. In this simulated bivariate dataset, one variable is overimputed and the results displayed. The second variable is either observed, in which case the confidence intervals are very small and the imputations (yellow) are very accurate, or the second variable is missing in which case this variable is being imputed simply from the mean and variance parameters, and the imputations (red) have a very large and encompassing spread. The circles represent the mean of all the imputations for that value. As the amount of missing information in a particular pattern of missingness increases, we expect the width of the confidence interval to increase. The color of the confidence interval reflects the percent of covariates observed in that pattern of missingness, as reflected in the legend at the bottom.

This graph can also be called from *R* as:

```
> output<-amelia(data=x)
> overimpute(data=x,output=output,var=3)
```

where `var` is the column number (or the name) of the variable you would like to

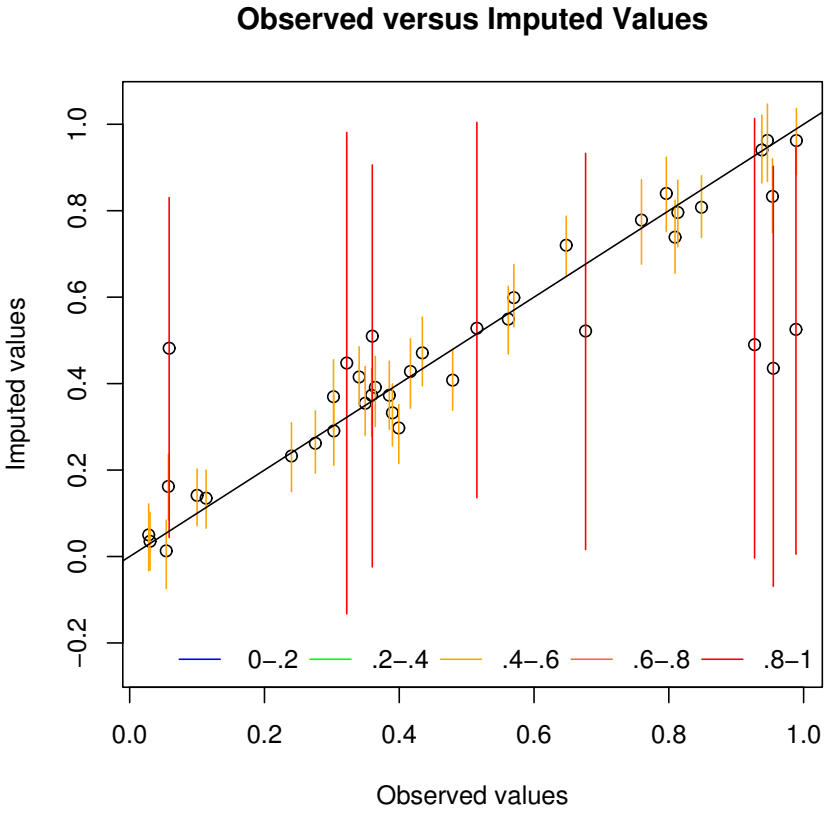

Figure 2: An example of the overimpute diagnostic graph. Here ninety percent confidence intervals are constructed that detail where an observed value would have been imputed had it been missing from the dataset, given the imputation model. The dots represent the mean imputation. Around ninety percent of these confidence intervals contain the $y = x$ line, which means that the true observed value falls within this range. The color of the line (as coded in the legend) represents the fraction of missing observations in the pattern of missingness for that observation. The yellow lines have more information (observed covariates) from which to impute the variable, and thus much tighter bounds.

produce the graph for.

9.3 Overdispersed Starting Values

If the data given to $\mathfrak{A}melia$ has a poorly behaved likelihood, the EM algorithm can have problems finding a global maximum of the likelihood surface and starting values can begin to effect imputations. Because the EM algorithm is deterministic, the point in the parameter space where you start it can impact where it ends, though this is irrelevant when the likelihood has only one mode. However, if the starting values of an EM chain are close to a local maximum, the algorithm may find this maximum, unaware that there is a global maximum farther away. To make sure that our imputations do not depend on our starting values, a good test is to run the EM algorithm from multiple, dispersed starting values and check their convergence. In a well behaved likelihood, we will see all of these chains converging to the same value, and reasonably conclude that this is the likely global maximum. On the other hand, we might see our EM chain converging to multiple locations. The algorithm may also wander around portions of the parameter space that are not fully identified, such as a ridge of equal likelihood, as would happen for example, if the same variablewere accidentally included in the imputation model twice.

$\mathfrak{A}melia$ includes a diagnostic to run the EM chain from multiple starting values that are overdispersed from the estimated maximum. The overdispersion diagnostic will display a graph of the paths of each chain. Since these chains move through spaces that are in an extremely high number of dimensions and can not be graphically displayed, the diagnostic reduces the dimensionality of the EM paths by showing the paths relative to the largest principle components of the final mode(s) that are reached. Users can choose between graphing the movement over the two largest principal components, or more simply the largest dimension with time (iteration number) on the x-axis. The number of EM chains can also be adjusted. Once the diagnostic draws the graph, the user can visually inspect the results to check that all chains convergence to the same point.

In one dimension, the diagnostic plots movement of the chain on the y-axis and time, in the form of the iteration number, on the x-axis. Figures 9.3 and 9.3 show two examples of one dimensional plots. The first shows a well behaved likelihood, as the starting values all converge to the same point. The black horizontal line is the point where $\mathfrak{A}melia$ converges when it uses the default method for choosing the starting values. The diagnostic takes the end point of this chain as the possible maximum and disperses the starting values away from it to see if the chain will ever finish at another mode. Figure 9.3 shows an example of how a diagnostic plot will look on a problematic likelihood surface. A few of the iterations are ending up in vastly different location in the parameter space. This can happen for a variety of reasons. In this example it is a result of having two highly collinear variables included in the imputation model. More generally, an unidentified imputation model will lead to non-unique ML estimates (see King (1989) for a more detailed discussion of identification and likelihoods).

This graph can also be called from R as:

```
> output<-amelia(data=x)
```

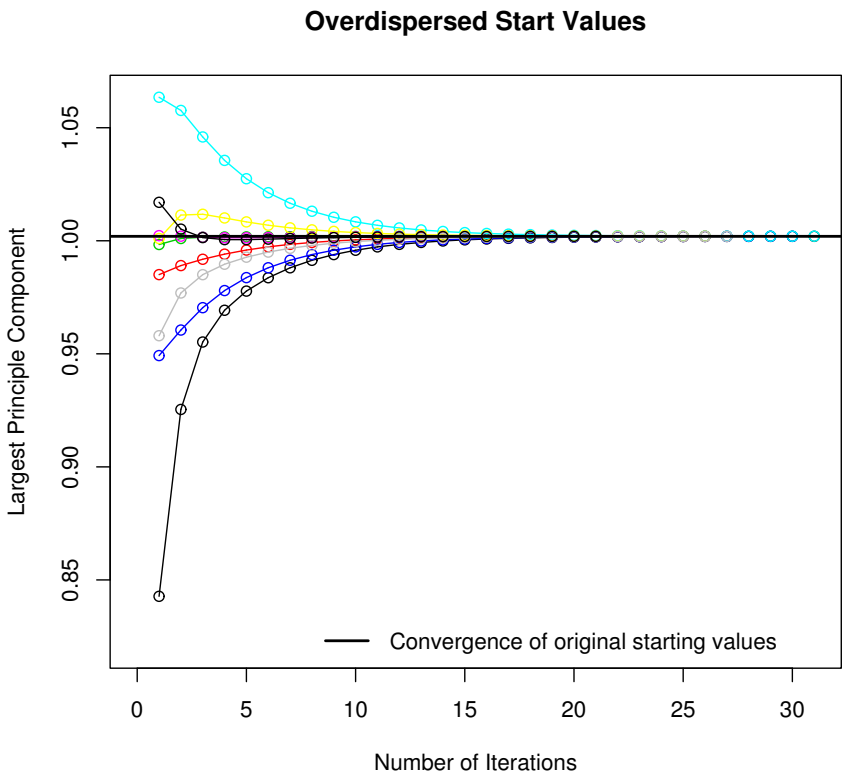

Figure 3: A plot from the overdispersion diagnostic where all EM chains are converging to the same mode, regardless of starting value. The y-axis represents movement in the (very high dimensional) parameter space, and the x-axis represents the iteration number of the chain.

```
> disperse(data=x,output=output,m=5,dims=1)
```

where `dims` can be set to a value of either 1 (default) or 2, for one dimensional and two dimensional graphs, respectively. The argument m sets the number of EM chains to run to create the graph. In *R* this routine can also be called before an$\mathfrak{A}$*melia* run has taken place, in which case output need not be declared, but any other $\mathfrak{A}$*melia* options may be set in the same way they would be declared for the `amelia` function. This allows you to check the likelihood surface *before* running the `amelia` function.

10 Sessions

Setting options for some large datasets may take some time, especially with the presence of many observational priors and non-linear variables. To ease this process, $\mathfrak{A}$*melia* has the ability to save and load sessions. A saved session includes all options set by the user in addition to any output information from $\mathfrak{A}$*melia* . Sessions are useful for setting large numbers of options over multiple instances of $\mathfrak{A}$*melia* or imputing the data multiple times with the same setup.

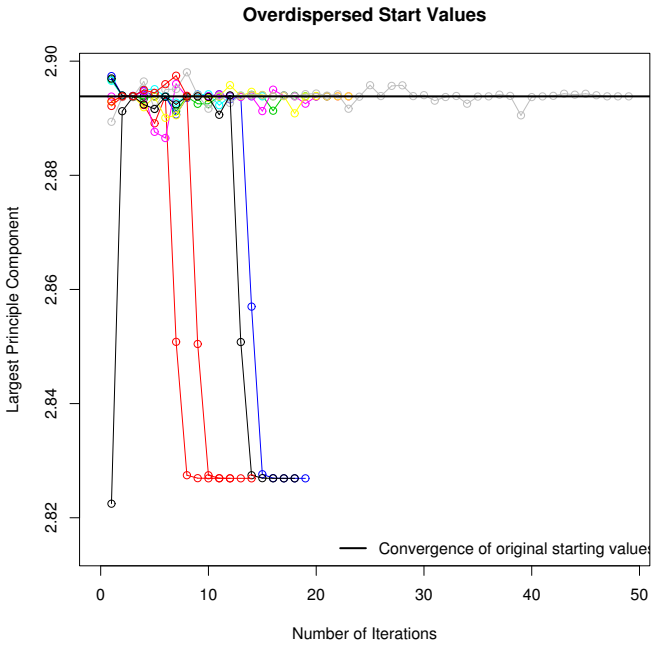

Figure 4: A problematic plot from the overdispersion diagnostic showing that EM chains are converging to one of two different modes, depending upon the starting value of the chain.

In AmeliaView, you can save a session by opening the "File" menu and selecting "Save Session." This will open a dialog that will ask for a filename (the file extension must be ".r"). To load a saved session, simply navigate to the "File" menu and select "Load Session" and locate the saved session file. Note that the data file referenced in the session file must be located in the same directory and have the same name when attempting to load the session. Upon loading, $\mathfrak{A}$*melia* configures all options to same values as the saved session.

In some cases you may wish to load a AmeliaView session at the R command line. To do so, use the `source` command to load the session file which will add a list named `amelia.list` to your working environment. You can use this list in the `arglist` options for the `amelia` function, which takes an $\mathfrak{A}$*melia* output or session list as its argument. An example of loading a session at the command line would be:

```
> source("am-session.R")
> amelia.out <- amelia(data=x,arglist=amelia.list)
```

In this example `am-session.R` is the session file and `x` is the dataset that corresponds to the session file. At the command line, you must load the data into *R* separately as sessions are not tied to specific datasets at the command line. When provided to $\mathfrak{A}$*melia* , the argument list will take precedence over other arguments. For example, specifying an identification variable with `idvars` while also using `arglist` will result in $\mathfrak{A}$*melia* ignoring the `idvars` provided by the user and refer to the idvars setting in `arglist`.

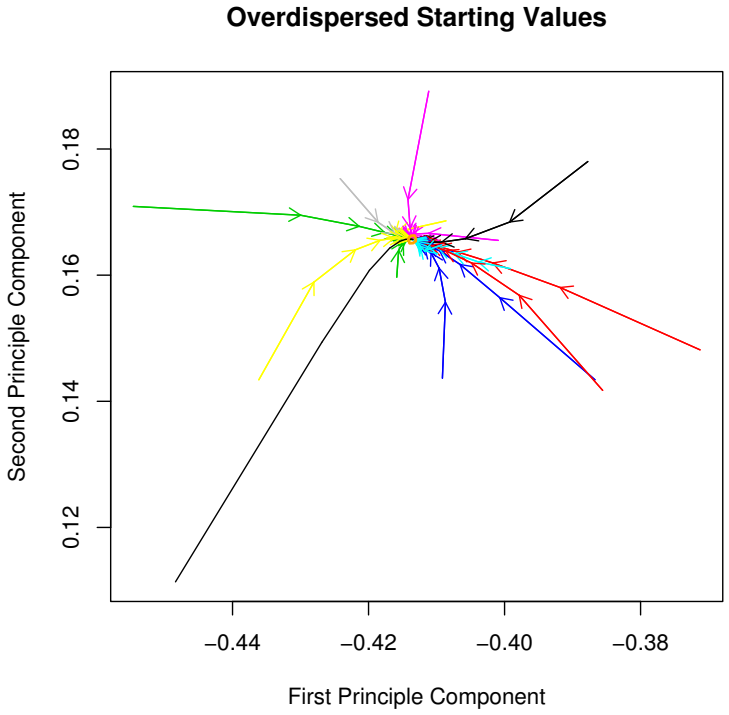

Figure 5: A alternate way to visualize the plot visualizing the parameter space in two dimensions using the first two principal components of the end points of the EM chains. The iteration number is no longer represented on the y-axis, although the distance between iterations is marked by the distance between arrowheads on each chain.

11 Error Checking and Reference Guide

In order to assure a smooth run of the algorithm, $\mathfrak{A}melia$ runs a check of all the inputs passed to it. If it finds an error, it will exit the function and present you with an error message and an error code. The vast majority of these checks keep your inputs from crashing $\mathfrak{A}melia$, but there are a few instances when you would want to turn off checking. In these cases, you can use the `incheck` argument to the `amelia()` function. Setting this option to `FALSE` will skip all error checking. Only run $\mathfrak{A}melia$ with this option set if you know that all your inputs are correct and you are sure you need to bypass a certain check.

11.1 Error Code Reference

Return Code	Error Message
1	Normal run of Amelia
2	One or more imputations ended with a non-invertible covariance matrix. Check for highly collinear variables.
3	One of the arguments points to an R variables that does not exists.
4	One variable in the data set is completely missing or only has one observation.
5	One of the variables you have referred to in an argument does not match any variable name in the data.
6	One of the column numbers in an argument is outside the range of the data.
7	The priors matrix is not a matrix.
8	The priors matrix is non-numeric.
9	There are priors specified for cells not in the data.
10	A variable marked for a square root transformation has negative values.
11	A variable marked for a logistic transformation has values outside of the 0-1 interval.
12	Confidence intervals on priors must be between 0 and 1.
13	Cannot set all the variables as identification variables.
14	The time series and cross sectional variables cannot be the same.
15	Only one time series variable can be specified.
16	Only one cross sectional variable can be specified.
17	The case priors must be in the form of a matrix.
18	The cross sectional variable must be set to use case priors.
19	The case priors have the wrong dimensions. They should be a kxk matrix, where is k is the number of cases.
20	Case prior values must be either 0, 1, 2, or 3.

Return Code	Error Message
21	The polynomials of time are longer than one variable.
22	The polynomials of time argument is not a number.
23	The polynomials of time argument is not an integer.
24	The polynomials of time argument is not between 0 and 3.
25	The polynomials of time argument is set without the time series variable being set.
26	The interact with the cross-section argument (`intercs`) is set without the cross-sectional variable being set.
28	There are too many cross sections to run $\mathfrak{A}melia$ with an interaction between the cross section and the polynomials of time.
29	One of the logical arguments is longer than one.
30	One of the logical arguments is not a logical (TRUE/FALSE) value.
31	One of the logical arguments is NULL, but it must be (TRUE/FALSE).
32	One of the variables is marked for more than one transformation.
33	One of the time series or cross sectional variables is marked for transformation, but these cannot be transformed.
34	The number of observations is too small to estimate the imputations for the number of parameters included. Drop some parameters or increase the empirical prior.
35	The tolerance parameter is less than or equal to zero. Your chains will never converge.
36	The number of categories in a variable marked as nominal is greater than one-third of the number of rows in the data. Make sure you have set this argument correctly.
37	There is factor variable that isn't marked as an identification, nominal, ordinal, or cross sectional variable. These are non-numeric variables and they cannot be imputed as anything other than a nominal or an ordinal.
38	There is a character variable in your data. Remove it from your data or convert it to a factor and set it accordingly (see error code 37).
39	There are no missing values in the data.
40	The lags argument is set without setting the time series variable.
41	The leads argument is set without setting the time series variable.
42	There is only one column of data. In order to impute, there has to be more than one variable.
43	There is a variable in your data that does not vary.
44	A variable marked as an ordinal has non-integer values.

Return Code	Error Message
45	The output filename is in a location that *Amelia* cannot write. Please check file permissions and try again.
46	The argument list you provided was invalid. The `arglist` argument only takes output lists from the `amelia` function or session lists from AmeliaView sessions.
47	The priors matrix has the wrong dimensions.
48	There are missing values in the priors matrix.
43	There are multiple priors set for one observation or variable..

12 Command Line Argument Reference

12.1 `amelia`: Multiple Imputation of Incomplete Data

Syntax

```
amelia(data,m=5,p2s=1,frontend=FALSE,idvars=NULL,logs=NULL,
       ts=NULL,cs=NULL,casepri=NULL,priors=NULL,empri=NULL,tolerance=0.0001,
       polytime=NULL,startvals=0,lags=NULL, leads=NULL,
       intercs=FALSE,archive=TRUE,sqrts=NULL,lgstc=NULL,
       noms=NULL,incheck=T,ords=NULL,collect=FALSE,
       outname="outdata",write.out=TRUE,arglist=NULL,keep.data=TRUE)
```

Arguments

- `data`: an incomplete dataset, organized into either a data frame or a matrix.
- `m`: the number of imputed datasets to create.
- `p2s`: an integer value taking either 0 for no screen output, 1 for normal screen printing of iteration numbers, and 2 for detailed screen output. See "Details" for specifics on output when p2s=2.
- `frontend`: a logical value used internally for the GUI.
- `idvars`: a vector of column numbers or column names that indicates identification variables. These will be dropped from the analysis but copied into the imputed datasets.
- `ts`: column number or variable name indicating the variable identifying time in time series data.
- `cs`: column number or variable name indicating the cross section variable.
- `polytime`: integer between 0 and 3 indicating what power of polynomial should be included in the imputation model to account for the effects of time. A setting of 0 would indicate constant levels, 1 would indicate linear time effects, 2 would indicate squared effects, and 3 would indicate cubic time effects.

- `intercs`: a logical variable indicating if the time effects of `polytime` should vary across the cross-section.
- `lags`: a vector of numbers or names indicating columns in the data that should have their lags included in the imputation model.
- `leads`: a vector of numbers or names indicating columns in the data that should have their leads (future values) included in the imputation model.
- `startvals`: starting values, 0 for the parameter matrix from listwise deletion, 1 for an identity matrix.
- `tolerance`: the convergence threshold for the EM algorithm.
- `logs`: a vector of column numbers or column names that refer to variables that require log-linear transformation.
- `sqrts`: a vector of numbers or names indicating columns in the data that should be transformed by a sqaure root function. Data in this column cannot be less than zero.
- `lgstc`: a vector of numbers or names indicating columns in the data that should be transformed by a logistic function for proportional data. Data in this column must be between 0 and 1.
- `noms`: a vector of numbers or names indicating columns in the data that are nominal variables.
- `ords`: a vector of numbers or names indicating columns in the data that should be treated as ordinal variables.
- `incheck`: a logical indicating whether or not the inputs to the function should be checked before running `amelia`. This should only be set to `FALSE` if you are extremely confident that your settings are non-problematic and you are trying to save computational time.
- `collect`: a logical value indicating whether or not the garbage collection frequency should be increased during the imputation model. Only set this to `TRUE` if you are experiencing memory issues as it can significantly slow down the imputation process.
- `outname`: a string indicating the prefix of the file to which Amelia will write the imputed datasets. You can also specify a path in front of the prefix if you do not wish your items stored in the working directory. The files will be written as .csv files.
- `write.out`: a logical value indicating whether or not you wish to have Amelia write your imputed datasets as comma-seperated value files. If `TRUE`, Amelia will use the `outname` arugment as the file prefix.

- `archive`: a logical variable indicating whether a replication archive should be saved. This archive includes all of the settings, the results of each imputation and some information about the convergence. The output will be saved as `'amarchive.R'` in your working directory.

- `arglist`: an output list from the `amelia` function or from a saved session from AmeliaView. Values from this list take precendent over any individually set arugments. See the Amelia manual for more information.

- `keep.data`:a logical value indicating whether or not to keep the imputed datasets after each imputation. Useful if the datasets are large and you wish to avoid keeping them in memory after they have been written to a file.

- `empri`:number indicating level of the empirical (or ridge) prior. This prior shinks the covariances of the data, but keeps the means and variances the same for problems of high missingness, small N's or large correlations among the variables. Should be kept small; a reasonable upper bound is around 10% of the rows of the data.

- `casepri`: indicator matrix of size *kxk* (where k is the number of cases) for the degree of similarity between two cases. For example, the [2,3] entry would indicate how similar cases 2 and 3 were. The indicators can be 0, 1, 2, or 3. Values should only appear in the upper triangle, as values in the lower triangle are ignored.

- `priors`: a four or five column matrix containing the priors for either individual missing observations or variable-wide missing values. See "Details" for more information.

- `autopri`: allows the EM chain to increase the empirical prior if the path strays into an nonpositive definite covariance matrix, up to a maximum empirical prior of the value of this argument times n, the number of observations. Must be between 0 and 1, and at zero this turns off this feature.

Details

Multiple imputation is a method for analyzing incomplete multivariate data. This function will take an incomplete dataset in either data frame or matrix form and return `m` imputed datasets with no missing values. The algorithm used first bootstraps a sample dataset with the same dimensions as the original data, estimates the sufficient statistics (with priors if specified) by EM, and then imputes the missing values of sample. It repeats this process `m` times to produces the m complete datasets where the observed values are the same and the unobserved values are drawn from their posterior distributions.

You can provide Amelia with informational priors about the missing observations in your data. To specify priors, pass a four or five column matrix to the `priors` argument with each row specifying a different priors as such:

```
one.prior <- c(row, column, mean,standard deviation)
```

or,

```
one.prior <- c(row, column, minimum, maximum, confidence).
```

So, in the first and second column of the priors matrix should be the row and column number of the prior being set. In the other columns should either be the mean and standard deviation of the prior, or a minimum, maximum and confidence level for the prior. You must specify your priors all as distributions or all as confidence ranges. Note that ranges are converted to distributions, so setting a confidence of 1 will generate an error.

Setting a priors for the missing values of an entire variable is done in the same manner as above, but inputing a `0` for the row instead of the row number. If priors are set for both the entire variable and an individual observation, the individual prior takes precedence.

If each imputation is taking a long time to converge, you can increase the empirical prior, `empri`. This value has the effect of smoothing out the likelihood surface so that the EM algorithm can more easily find the maximum. It should be kept as low as possible and only used if needed.

Amelia assumes the data is distributed multivariate normal. There are a number of variables that can break this assumption. Usually, though, a transformation can make any variable roughly continuous and unbounded. We have included a number of commonly needed transformations for data. Note that the data will not be transformed in the output datasets and the transformation is simply useful for climbing the likelihood.

Please refer to the Amelia manual for more information on the function or the options.

Output

A list containing the imputed datasets in objects 1 through `m`. Thus, you can refer to any of the datasets by referencing `output[[i]]`, where `i` is the number of the dataset you wish to reference.

These datasets will be returned in the same format which you passed them. For example, if you passed a data frame to `amelia` you will have `m` data frames in the output list. If you passed a matrix, you will have `m` matrices in the output.

Other objects in the list:

- `code`: return code for the function. 0 indicates a successful run of Amelia. Other codes refer to various problems in data or settings. Please refer to the error message and the Amelia manual for help with errors.
- `message`: error message. Only appears if return code is not 0.
- `amelia.args`: list of the arguments used in the imputation along with a few diagnostics on each imputation.
- `thetas`:a matrix of the output parameter matrices used to generate the imputed datasets.

13 AmeliaView Menu Guide

13.1 Step 1 - Input

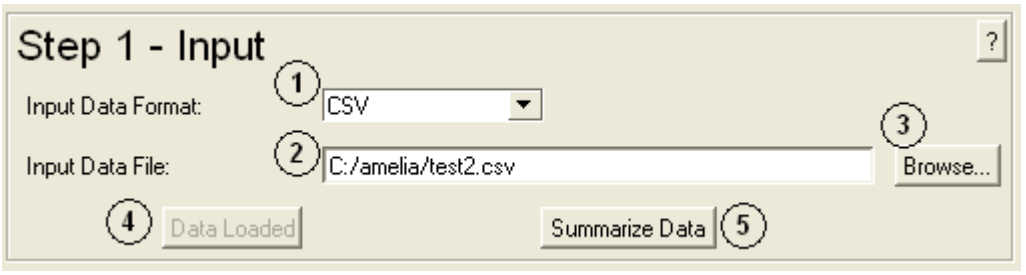

Figure 6: Detail for step 1 on the front page of AmeliaView.

1. **Input Data Format** - Choose the format for your dataset. The format you pick will be the default format that is shown when you open the "Browse" dialog. Currently, *Amelia* supports five different file formats: Comma-Separated Values (.CSV), Tab-Delimited Text (.TXT), Stata v.5-8 (.DTA), SPSS (.DAT), and SAS Transport (.XPORT).

2. **Input Data File** - Enter the location of your dataset. If your file is located in a high level directory, it might be you are trying to access for more information.

3. **Summarize Data** - View plots and summary statistics for the individual variables. This button will bring up a dialog box with a list of variables. Clicking on each variable will display the summary statistics on the right. Below these statistics, there is a "Plot Variable" button, which will show a histogram of the variable. For data that are string or character based, AmeliaView will not show summary statistics or plot histograms.

13.2 Step 2 - Options

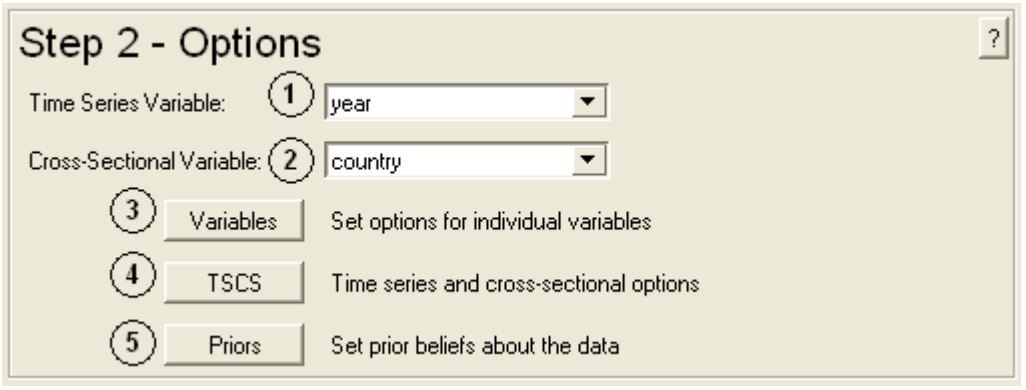

Figure 7: Detail for step 2 on the front page of AmeliaView.

1. **Time Series Variable** - Choose the variable that indexes time in the dataset. If there is no time series component in your data, set it to "(none)." You must set this option in order to access the Time Series Cross Sectional options dialog.

2. **Cross Sectional Variable** - Choose the variable that indexes the cross-section. You must set this in order to access the "Set Case Priors" in the "Priors" dialog.

3. **Variables** - Becomes available after you load the data. See 13.2.1 for more information.

4. **TSCS** - Becomes available after you set the Time Series variable. See 13.2.2 for more information.

5. **Priors** - Becomes available after you load the data. See 13.2.3 for more information.

13.2.1 Variables Dialog

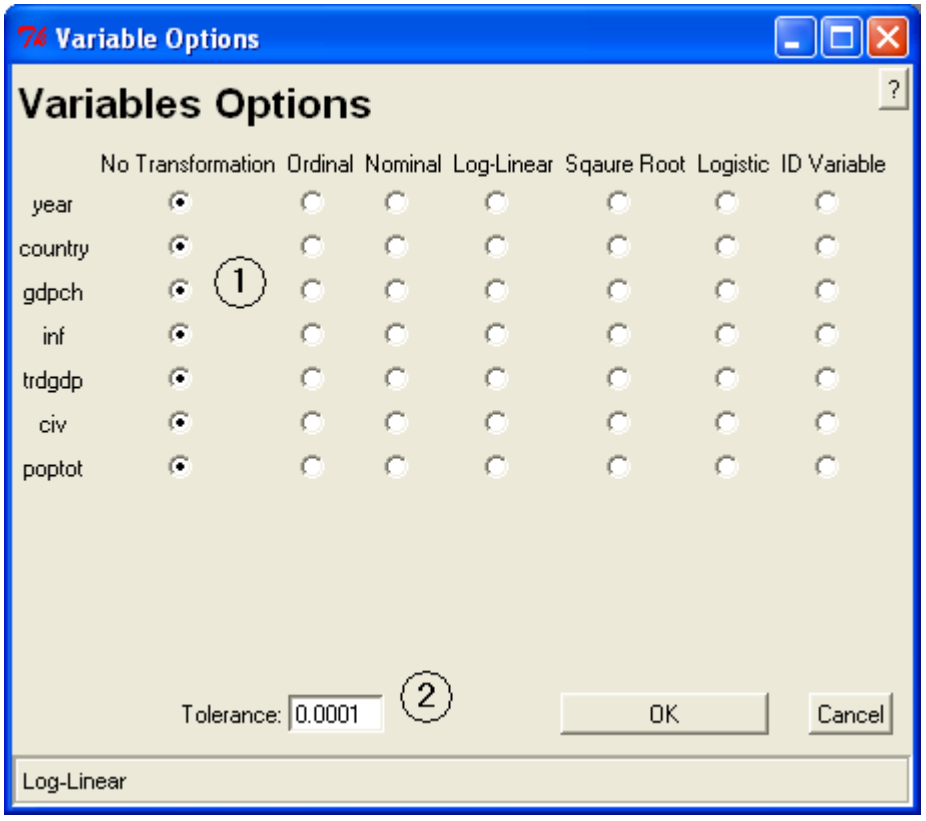

Figure 8: Detail for Variable Options dialog.

1. **Variable Transformations** - Choose the transformation that best tailors the variable to the multivariate normal, if appropriate. See 7.2 on Transformations to see how each transformation is useful. You can also choose whether or not

the variable is an identification (ID) variable. If so, it will be left out of the imputation model, but will remain in the imputed datasets. This is useful for variables that have no explanatory power like extra case identifiers.

2. **Tolerance** - Adjust the level of tolerance that *Amelia* uses to check convergence of the EM algorithm. In very large datasets, if your imputation chains run a long time without converging, increasing the tolerance will allow a lower threshold to judge convergence and end chains after fewer iterations.

13.2.2 Time Series Cross Sectional Dialog

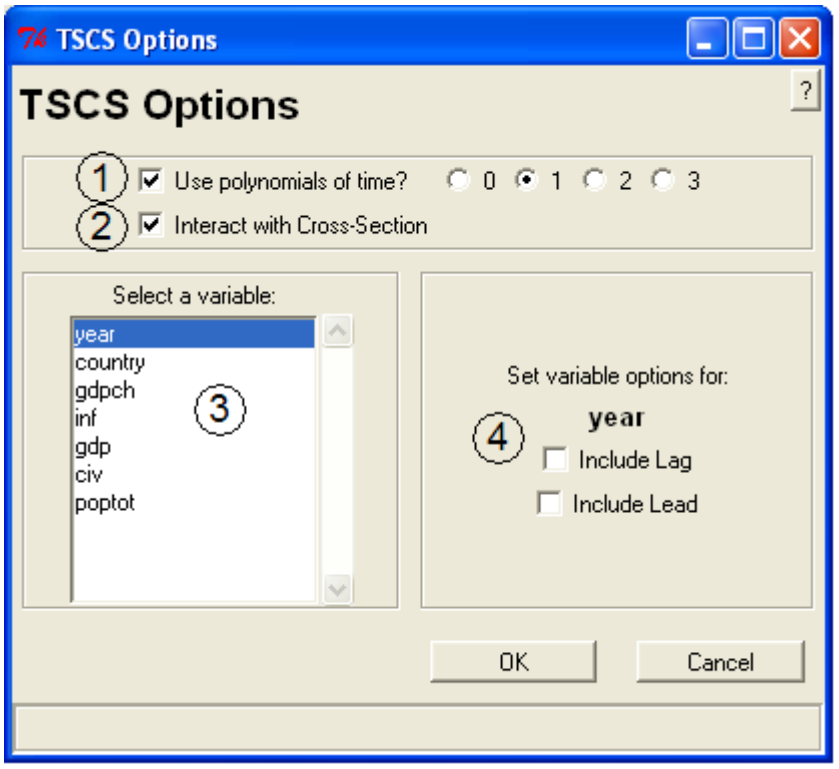

Figure 9: Detail for Time-Series-Cross-Section Options dialog.

1. **Polynomials of Time** - This option, if activated, will have *Amelia* use trends of time as a additional condition for fitting the missing data. The higher the level of polynomial will allow more variation in the trend structure, yet it will take more degrees of freedom to estimate.

2. **Interact with Cross-Section** - Interacting this with the cross section is way of allowing the trend of time to vary across cases as well. Using a 0 level polynomial and interacting with the cross section is the equivalent of using a fixed effects. For more information see 7.4 above.

3. **Variable Listbox** - Choose the variables whose lag or lead you would like to include in the imputation model.

4. **Lag Settings** - Choose to include lags and leads in the data set to handle the effects of time. See 7.4 above.

13.2.3 Priors Dialog

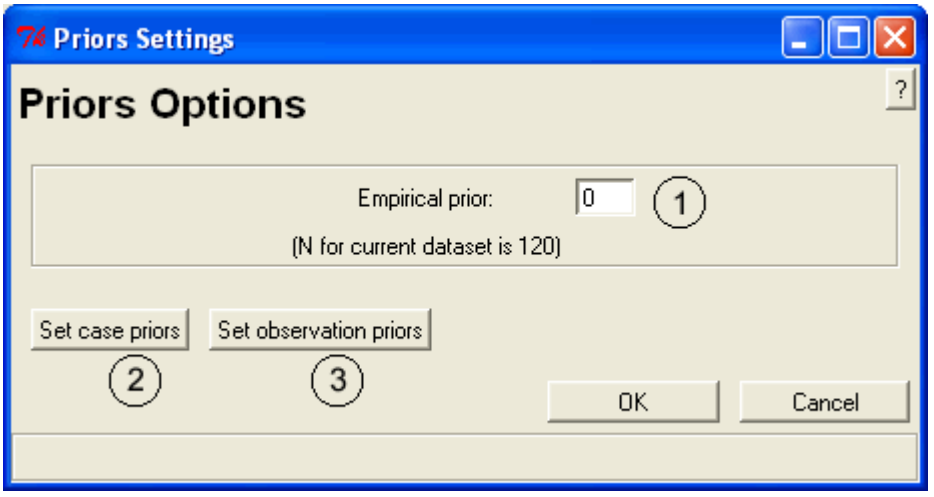

Figure 10: Detail for Priors Options dialog.

1. **Empirical Prior** - A prior that adds observations to your data in order to shrink the covariances. A useful place to start is around 0.5% of the total number of observations in the dataset.

2. **Set Case Priors** - Tell 𝔄*melia* which cases are similar. See 13.2.4 for more details about case priors.

3. **Set Observational Priors** - Set prior beliefs about ranges for individual missing observations. For more information about observational priors, see 8.2.

13.2.4 Case Priors

1. **Case Names** - The row and column name for each button indicate the relationship that the button controls. For instance, the top-left button in the "Burkina Faso" row and the "Burundi" column; thus, the setting on this button will indicate the strength of the similarity prior between these two cross-sections.

2. **Buttons** - Toggling the buttons will change the setting for each case. The higher the number on the button, the stronger the indicated relationship is believed to be. Find the column of the first country and then follow it down to the row of the similar country. Press the button to raise the level similarity from 0 to 1 to 2 to 3. You can reset the case priors by hitting the "Reset All" button in the top left corner.

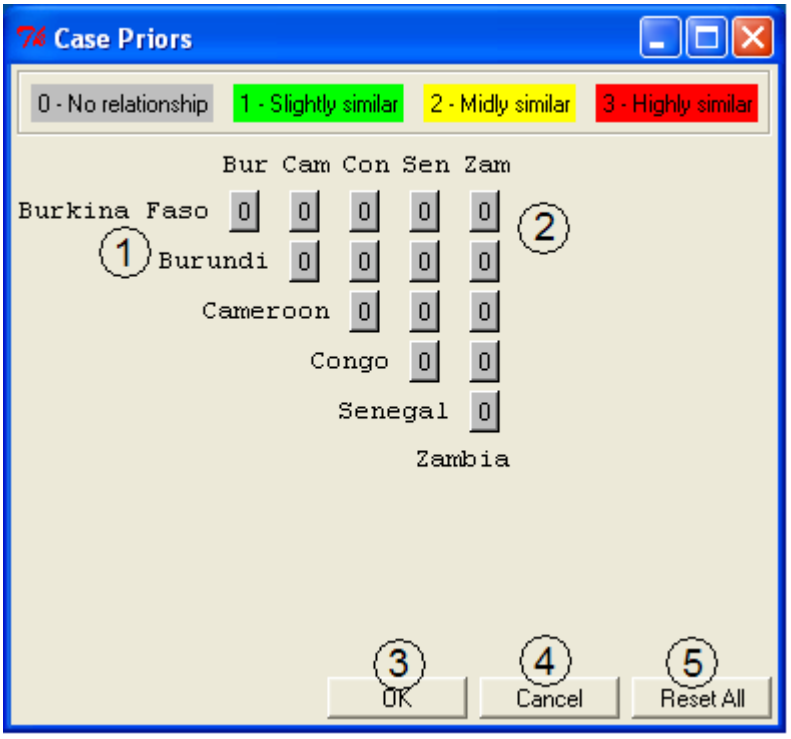

Figure 11: Detail for Case Priors dialog.

3. **OK** - Close the window and save any changes.
4. **Cancel** - Close the window and discard any changes.
5. **Reset All** - Sets all case priors to the default 0 setting.

13.2.5 Observational Priors

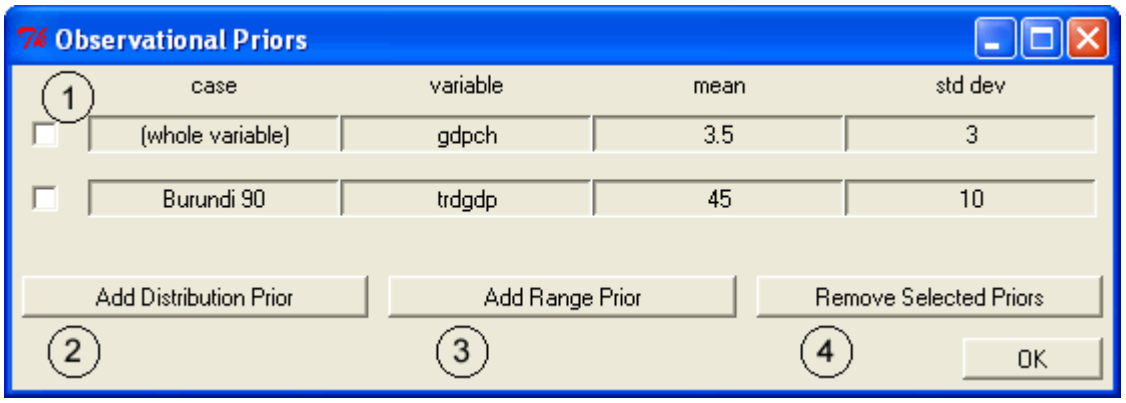

Figure 12: Detail for Observational Priors dialog

1. **Current Priors** - A list of current priors in distributional form, with the variable and case name.

2. **Add Distributional Prior** - Add a prior belief about an observation or an entire variable with a mean and standard deviation about the missing values.

3. **Add Range Prior** - Add a prior belief about an observation or an entire variable with a range and a confidence level.

4. **Remove Selected Priors** - This will remove any of the current priors selected with the check box.

13.2.6 Add Distribution Prior

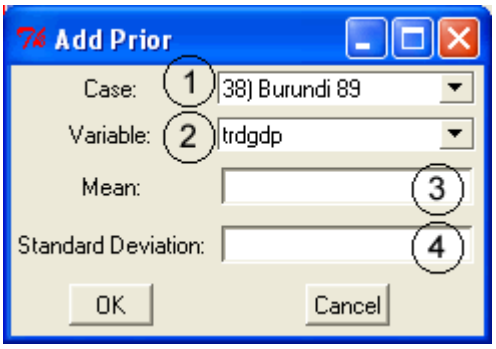

Figure 13: Detail for Add Distributional Prior dialog

1. **Case** - Select the case name or number you wish to set the prior about. You can also choose to make the prior for the entire variable. The case names are generated from the row name of the observation, the value of the cross-section variable of the observation and the value of the time series variable of the observation.

2. **Variable** - The variable associated with the prior you would like specify. The list provided only shows the missing variables for the currently selected observation.

3. **Mean** - The mean value of the prior. The textbox will not accept letters or out of place punctuation.

4. **Standard Deviation** - The standard deviation of the prior. The textbox will only accept positive non-zero values.

13.2.7 Add Range Prior

1. **Case** - Select the case name or number you wish to set the prior about. You can also choose to make the prior for the entire variable. The case names are generated from the row name of the observation, the value of the cross-section variable of the observation and the value of the time series variable of the observation.

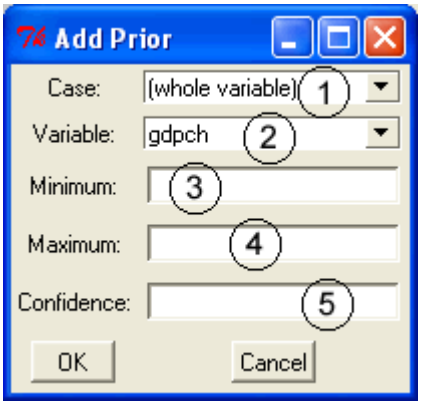

Figure 14: Detail for Add Range Prior dialog

2. **Variable** - The variable associated with the prior you would like specify. The list provided only shows the missing variables for the currently selected observation.

3. **Minimum** - The minimum value of the prior. The textbox will not accept letters or out of place punctuation.

4. **Maximum** - The maximum value of the prior. The textbox will not accept letters or out of place punctuation.

5. **Confidence** - The confidence level of the prior. This should be between 0 and 1, non-inclusive. This value represents how certain your priors are. This value cannot be 1, even if you are absolutely certain of a give range. This is used to convert the range into an appropriate distributional prior.

13.3 Step 3 - Output

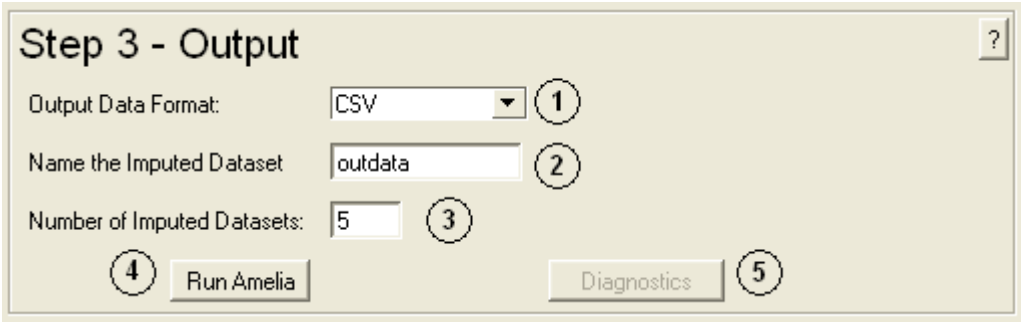

Figure 15: Detail for step 3 on the front page of AmeliaView.

1. **Output Data Format** - Choose the format of output data. If you would like to not save any output data sets (if you wanted, for instance, to simply look at diagnostics), set this option to "(no save)." Currently, you can save the output data as: Comma Separated Values (.CSV), Tab Delimited Text (.TXT), or Stata (.DTA).

2. **Name of Imputed Datasets** - Enter the prefix for the output data files. If you set this to "mydata", your output files will be `mydata1.csv, mydata2.csv...` etc. Try to keep this name short as some operating systems have a difficult time reading long filenames.

3. **Number of Imputed Datasets** - Set the number of imputations you would like. In most cases, 5 will be enough to make accurate predictions about the means and variances.

4. **Run Amelia** - Runs the *Amelia* procedure on the input data. A dialog will open marking the progress of *Amelia*. Once it is finished, it will tell you that you can close the dialog. If an error message appears, follow its instructions; this usually involves closing the dialog, resetting the options, and running the procedure again.

5. **Diagnostics** - Post-imputation diagnostics. The only currently available graph compares the densities of the observed data to the mean imputation across the m imputed datasets.

13.3.1 Diagnostics Dialog

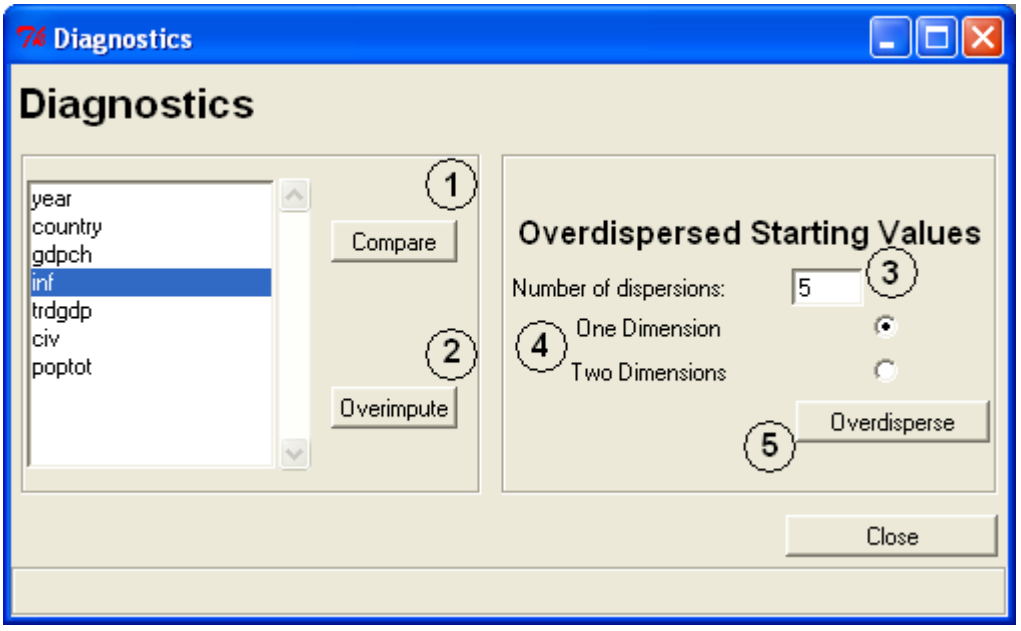

Figure 16: Detail for Diagnostics dialog.

1. **Compare Plots** - This will display the relative densities of the observed (red) and imputed (black) data. The density of the imputed values are the average imputations across all of the imputed datasets.

2. **Overimpute** - This will run *Amelia* on the full data with one cell of the chosen variable artificially set to missing and then check the result of that

imputation against the truth. The resulting plot will plot average imputations against true values along with 90% confidence intervals. These are plotted over a $y = x$ line for visual inspection of the imputation model.

3. **Number of overdispersions** - When running the overdispersion diagnostic, you need to run the imputation algorithm from several overdispersed starting points in order to get a clear idea of how the chain are converging. Enter the number of imputations here.

4. **Number of dimensions** - The overdispersion diagnostic must reduce the dimensionality of the paths of the imputation algorithm to either one or two dimensions due to graphical restraints.

5. **Overdisperse** - Run overdispersion diagnostic to visually inspect the convergence of the $\mathfrak{A}$*melia* algorithm from multiple start values that are drawn randomly.

References

Abayomi, Kobi, Andrew Gelman and Marc Levy. 2005. "Diagnostics for Multivariate Imputations." *Working Paper* .

King, Gary. 1995. "Replication, Replication." *PS: Political Science and Politics* 28(3, September):443–499. http://gking.harvard.edu/files/abs/replication-abs.shtml.

King, Gary, James Honaker, Anne Joseph and Kenneth Scheve. 2001. "Analyzing Incomplete Political Science Data: An Alternative Algorithm for Multiple Imputation." *American Political Science Review* 95(1, March):49–69. http://gking.harvard.edu/files/abs/evil-abs.shtml.

King, Gary, Michael Tomz and Jason Wittenberg. 2000. "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science* 44(2, April):341–355. http://gking.harvard.edu/files/abs/making-abs.shtml.

Schafer, Joseph L. 1997. *Analysis of incomplete multivariate data*. London: Chapman & Hall.