# AMELIA II: A Program for Missing Data

James Honaker, Gary King, and Matthew Blackwell

Version 1.2-11
July 10, 2009

## Contents

# 1 Introduction

Missing data is a perennial problem in social science data. Respondents do not answer every question, countries do not collect statistics in every year and, most unfortunately, researchers do not always have the resources to collect every piece of available data. Most statistical analysis methods, however, assume the absence of missing data. Amelia II allows users to impute ("fill in" or rectangularize) incomplete data sets so that analyses which require complete observations can appropriately use all the information present in a dataset with missingness, and avoid the biases, inefficiencies, and incorrect uncertainty estimates that can result from dropping all partially observed observations from the analysis.

Amelia II performs *multiple imputation*, a general-purpose approach to data with missing values. Multiple imputation has been shown to reduce bias and increase efficiency compared to listwise deletion. Furthermore, ad-hoc methods of imputation, such as mean imputation, can lead to serious biases in variances and covariances. Unfortunately, creating multiple imputations can be a burdensome process due to technical nature of algorithms involved. Ameliaprovides users with a simple way to create an imputation model, implement it, and check its fit using diagnostics.

The Amelia II program goes several significant steps beyond the capabilities of the first version of Amelia (Honaker, Joseph, King, Scheve and Singh., 1998-2002). For one, the bootstrap-based EMB algorithm included in Amelia 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 Amelia II so that it virtually never crashes — which to our knowledge makes it unique among all existing multiple imputation software — and is much faster than the alternatives too. Amelia 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. In addition to all of this, Amelia II provides many diagnostic functions that help users check the validity of their imputation model. This software implements the ideas developed in Honaker and King (2009).

# 2 What Amelia 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 Amelia II's EMB algorithm, you can apply whatever statistical method you would have used if there had been no missing values in each of the $m$ data sets, and use a simple procedure, described below, to combine the results[1]. 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

[1](You can combine the results automatically by doing your data analyses within Zelig for R, or within Clarify for Stata; see `http://gking.harvard.edu/stats.shtml`.

𝔸melia 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.

## 2.1 Assumptions

The imputation model in 𝔸melia 𝕀𝕀 assumes that the complete data (that is, both observed and unobserved) are multivariate normal. If we denote the $(n \times k)$ dataset as $D$ (with observed part $D^{\text{obs}}$ and unobserved part $D^{\text{mis}}$), then this assumption is

$$D \sim \mathcal{N}_k(\mu, \Sigma), \tag{1}$$

which states that $D$ has a multivariate normal distribution with mean vector $\mu$ and covariance matrix $\Sigma$. The multivariate normal distribution is often a crude approximation to the true distribution of the data, yet there is evidence that this model works as well as other, more complicated models even in the face of categorical or mixed data (see Schafer, 1997; Schafer and Olsen, 1998). Furthermore, transformations of the data can often make this normality assumption more plausible (see 5.3 for more information on how to implement this in 𝔸melia).

The essential problem of imputation is that we only observe $D^{\text{obs}}$, not all of $D$. In order to gain traction, we need to make the usual assumption in multiple imputation that the data are *missing at random* (MAR). This assumption means that the pattern of missingness only depends on the observed data $D^{\text{obs}}$, not the unobserved data $D^{\text{mis}}$. Let $M$ to be the missingness matrix, with cells $m_{ij} = 1$ if $d_{ij} \in D^{\text{mis}}$ and $m_{ij} = 0$ otherwise. Put simply, $M$ is a matrix that indicates whether or not a cell is missing in the data. With this, we can define the MAR assumption as

$$p(M|D) = p(M|D^{\text{obs}}). \tag{2}$$

Note that MAR includes the case when missing values are created randomly by, say, coin flips, but it also includes many more sophisticated missingness models. When missingness is not dependent on the data at all, we say that the data are *missing completely at random* (MCAR). 𝔸melia requires both the multivariate normality and the MAR assumption (or the simpler special case of MCAR). Note that the MAR assumption can be made more plausible by including more information into the imputation model.

## 2.2 Algorithm

In multiple imputation, we are concerned with the complete-data parameters, $\theta = (\mu, \Sigma)$. When writing down a model of the data, it is clear that our observed data is actually $D^{\text{obs}}$ and $M$, the missingness matrix. Thus, the likelihood of our observed

data is $p(D^{\text{obs}}, M|\theta)$. Using the MAR assumption[2], we can break this up,

$$p(D^{\text{obs}}, M|\theta) = p(M|D^{\text{obs}})p(D^{\text{obs}}|\theta). \tag{3}$$

As we only care about inference on the complete data parameters, we can write the likelihood as

$$L(\theta|D^{\text{obs}}) \propto p(D^{\text{obs}}|\theta), \tag{4}$$

which we can rewrite using the law of iterated expectations as

$$p(D^{\text{obs}}|\theta) = \int p(D|\theta)dD^{\text{mis}}. \tag{5}$$

With this likelihood and a flat prior on $\theta$, we can see that the posterior is

$$p(\theta|D^{\text{obs}}) \propto p(D^{\text{obs}}|\theta) = \int p(D|\theta)dD^{\text{mis}}. \tag{6}$$

The main computational difficulty in the analysis of incomplete data is taking draws from this posterior. The EM algorithm (Dempster, Laird and Rubin, 1977) is a simple computational approach to finding the mode of the posterior. Our EMB algorithm combines the classic EM algorithm with a bootstrap approach to take draws from this posterior. For each draw, we bootstrap the data to simulate estimation uncertainty and then run the EM algorithm to find the mode of the posterior for the bootstrapped data, which gives us fundamental uncertainty too (see Honaker and King (2009) for details of the EMB algorithm).

Once we have draws of the posterior of the complete-data parameters, we make imputations by drawing values of $D^{\text{mis}}$ from its distribution conditional on $D^{\text{obs}}$ and the draws of $\theta$, which is a linear regression with parameters that can be calculated directly from $\theta$.

## 2.3 Analysis

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, $q_j$ $(j = 1, \ldots, m)$:

$$\bar{q} = \frac{1}{m}\sum_{j=1}^{m} q_j. \tag{7}$$

[2]There is an additional assumption hidden here that $M$ does not depend on the complete-data parameters.

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

$$SE(q)^2 = \frac{1}{m}\sum_{j=1}^{m} SE(q_j)^2 + S_q^2(1 + 1/m). \tag{8}$$

# 3 Versions of Amelia

Two versions of Amelia II are available, each with its own advantages and drawbacks, but both of which use the same underyling code. First, Amelia II exists as a package for the R statistical software package. Users can utilize their knowledge of the R language to run Amelia II at the command line or to create scripts that will run Amelia II and preserve the commands for future use. Alternatively, you may prefer AmeliaView, where a interactive Graphical User Interface (GUI) allows you to set options and run Amelia without any knowledge of the R programming language.

Both versions of Amelia II are available on the Windows, Mac OS X, and Linux platforms and Amelia II for R runs in any environment that R can. All versions of Amelia require the R software, which is freely available at `http://www.r-project.org/`.

# 4 Installation and Updates

Before installing Amelia II, you must have installed R version 2.1.0 or higher, which is freely available at `http://www.r-project.org/`.

To install the Amelia package on any platform, simply type the following at the R command prompt,

```
> install.packages("Amelia")
```

and R will automatically install the package to your system from CRAN. If you wish to use the most current beta version of Amelia feel free to install the test version,

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

In order to keep your copy of Amelia completely up to date, you should use the command

```
> update.packages()
```

## 4.1 Windows — AmeliaView

To install a standalone version of 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 Amelia II. If you have installed R with the default options, Amelia 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-2.9.0) and contains a subdirectory named `bin`. The installer will also put shortcuts on your Desktop and Start Menu.

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:

```
> library(Amelia)
> AmeliaView()
```

## 4.2 Linux (local installation)

Installing Amelia on a Linux system is slightly more complicated due to user permissions. If you are running R with root access, you can simply run the above installation procedure. If you do not have root access, you can install Amelia to a local library. First, create a local directory to house the packages,

```
w4:mblackwell [~]: mkdir ~/myrlibrary
```

and then, in an R session, install the package directing R to this location:

```
> install.packages("Amelia", lib = "~/myrlibrary")
```

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 A User's Guide

## 5.1 Data and Initial Results

We now demonstrate how to use Amelia using data from Milner and Kubota (2005) which studies the effect of democracy on trade policy. For the purposes of this user's

guide, we will use a subset restricted to nine developing countries in Asia from 1980 to 1999[3]. This dataset includes 9 variables: year (`year`), country (`country`), average tariff rates (`tariff`), Polity IV score[4] (`polity`), total population (`pop`), gross domestic product per capita (`gdp.pc`), gross international reserves (`intresmi`), a dummy variable for if the country had signed an IMF agreement in that year (`signed`), a measure of financial openness (`fivop`), and a measure of US hegemony[5] (`usheg`). These variables correspond to the variables used in the analysis model of Milner and Kubota (2005) in table 2.

We first load the Amelia and the data:

```
> require(Amelia)

##
## Amelia II: Multiple Imputation
## (Version 1.2-11, built: 2009-07-10)
## Copyright (C) 2005-2009 James Honaker, Gary King and Matthew Blackwell
## Refer to http://gking.harvard.edu/amelia/ for more information
##

> data(freetrade)
```

We can check the summary statistics of the data to see that there is missingness on many of the variables:

```
> summary(freetrade)

      year        country             tariff          polity
 Min.   :1981   Length:171         Min.   :  7.1   Min.   :-8.00
 1st Qu.:1985   Class :character   1st Qu.: 16.3   1st Qu.:-2.00
 Median :1990   Mode  :character   Median : 25.2   Median : 5.00
 Mean   :1990                      Mean   : 31.7   Mean   : 2.91
 3rd Qu.:1995                      3rd Qu.: 40.8   3rd Qu.: 8.00
 Max.   :1999                      Max.   :100.0   Max.   : 9.00
                                   NA's   : 58.0   NA's   : 2.00
      pop               gdp.pc          intresmi          signed
 Min.   :1.41e+07   Min.   :  150   Min.   : 0.904   Min.   :0.000
 1st Qu.:1.97e+07   1st Qu.:  420   1st Qu.: 2.223   1st Qu.:0.000
 Median :5.28e+07   Median :  814   Median : 3.182   Median :0.000
 Mean   :1.50e+08   Mean   : 1867   Mean   : 3.375   Mean   :0.155
 3rd Qu.:1.21e+08   3rd Qu.: 2463   3rd Qu.: 4.406   3rd Qu.:0.000
 Max.   :9.98e+08   Max.   :12086   Max.   : 7.935   Max.   :1.000
                                    NA's   :13.000   NA's   :3.000
```

[3]We have artificially added some missingness to these data for presentational purposes. You can access the original data at `http://www.princeton.edu/~hmilner/Research.htm`

[4]The Polity score is a number between -10 and 10 indicating how democratic a country is. A fully autocratic country would be a -10 while a fully democratic country would be 1 10.

[5]This measure of US hegemony is the US imports and exports as a percent of the world total imports and exports.

```
     fiveop          usheg
 Min.   :12.3   Min.   :0.256
 1st Qu.:12.5   1st Qu.:0.262
 Median :12.6   Median :0.276
 Mean   :12.7   Mean   :0.276
 3rd Qu.:13.2   3rd Qu.:0.289
 Max.   :13.2   Max.   :0.308
 NA's   :18.0
```

In the presence of missing data, most statistical packages use *listwise deletion*, which removes any row that contains a missing value from the analysis. Using the base model of Milner and Kubota (2005) table 2, we run a simple linear model in R, which uses listwise deletion:

```
> summary(lm(tariff ~ polity + pop + gdp.pc + year + country,
+     data = freetrade))

Call:
lm(formula = tariff ~ polity + pop + gdp.pc + year + country,
    data = freetrade)

Residuals:
     Min       1Q   Median       3Q      Max
-30.7640  -3.2595   0.0868   2.5983  18.3097

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         1.97e+03   4.02e+02    4.91  3.6e-06
polity             -1.37e-01   1.82e-01   -0.75     0.45
pop                -2.02e-07   2.54e-08   -7.95  3.2e-12
gdp.pc              6.10e-04   7.44e-04    0.82     0.41
year               -8.71e-01   2.08e-01   -4.18  6.4e-05
countryIndonesia   -1.82e+02   1.86e+01   -9.82  3.0e-16
countryKorea       -2.20e+02   2.08e+01  -10.61  < 2e-16
countryMalaysia    -2.25e+02   2.17e+01  -10.34  < 2e-16
countryNepal       -2.16e+02   2.25e+01   -9.63  7.7e-16
countryPakistan    -1.55e+02   1.98e+01   -7.84  5.6e-12
countryPhilippines -2.04e+02   2.09e+01   -9.77  3.7e-16
countrySriLanka    -2.09e+02   2.21e+01   -9.46  1.8e-15
countryThailand    -1.96e+02   2.10e+01   -9.36  3.0e-15

Residual standard error: 6.22 on 98 degrees of freedom
  (60 observations deleted due to missingness)
Multiple R-squared: 0.925,        Adjusted R-squared: 0.915
F-statistic:  100 on 12 and 98 DF,  p-value: <2e-16
```

Note that 60 of the 171 original observations are deleted due to missingness. Most of these observations, however, have information in them and multiple imputation will help us retrieve that information and make better inferences.

## 5.2 Multiple Imputation

When performing multiple imputation, the first step is to identify the variables to include in the model. It is crucial to include at least as much information as in the analysis model. That is, any variable that will be in the analysis model should also be in the imputation model. In fact, it is often useful to add more information. Since imputation is predictive, any variables that would increase predictive power should be included in the model, even if including them in the analysis model would produce bias (such as for post-treatment variables). In our case, we include all the variables in `freetrade` in the imputation model, even though we our analysis model focuses on `polity`, `pop` and `gdp.pc`[6].

To create multiple imputations in Amelia, we can simply run

```
> a.out <- amelia(freetrade, m = 5, ts = "year", cs = "country")

-- Imputation 1 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16

-- Imputation 2 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14

-- Imputation 3 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

-- Imputation 4 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21

-- Imputation 5 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17

> a.out

Amelia output with 5 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  16
```

[6]Note that this specification does not utilize time or spatial data yet. The `ts` and `cs` arguments only have force when we also include `polytime` or `intercs`, discussed in section 5.5

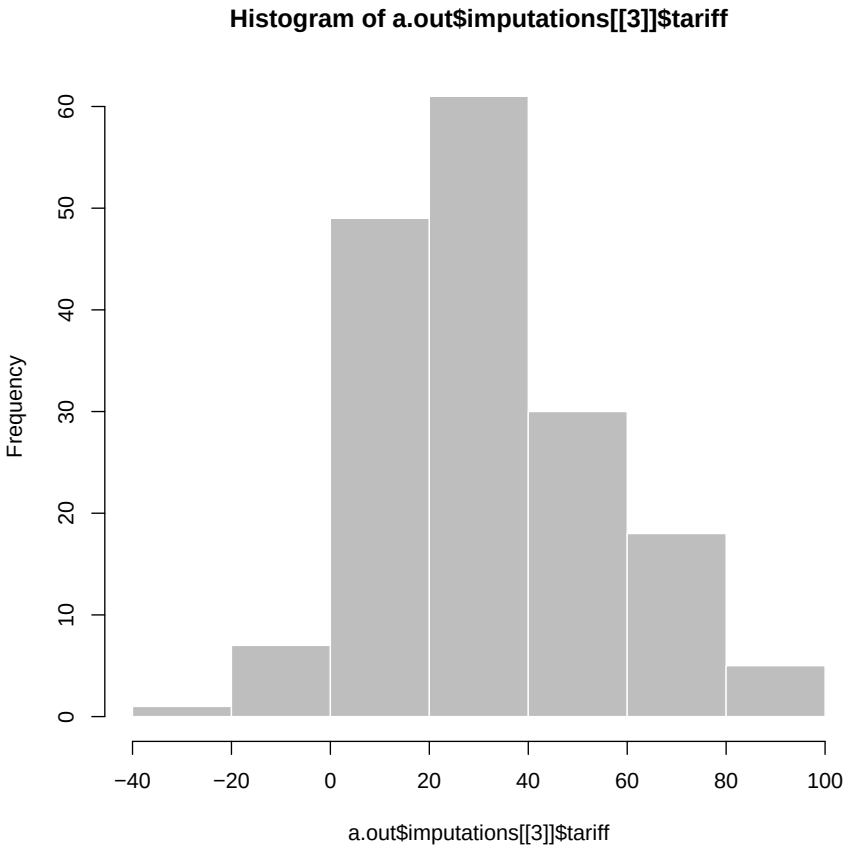


Figure 1: Histogram of the `tariff` variable from the 3rd imputated dataset.

```
Imputation 2:  14
Imputation 3:  15
Imputation 4:  21
Imputation 5:  17
```

The output gives some information about how the algorithm ran. Each of the imputed datasets is now in the list `a.out$imputations`. Thus, we could plot a histogram of the `tariff` variable from the 3rd imputation,

```
> hist(a.out$imputations[[3]]$tariff, col = "grey", border = "white")
```

### 5.2.1 Saving imputed datasets

If you need to save your imputed datasets, you can either save the output from `amelia`,

```
> save(a.out, file = "imputations.RData")
```

In addition, you can save each of the imputed datasets to its own file using the `write.amelia` command,

```
> write.amelia(obj=a.out, file.stem = "outdata")
```

This will create one comma-separated value file for each imputed dataset in the following manner:

```
outdata1.csv
outdata2.csv
outdata3.csv
outdata4.csv
outdata5.csv
```

The `write.amelia` function can also save files in tab-delimited and Stata (`.dta`) file formats. For instance, to save Stata files, simply change the `format` argument to `"dta"`,

```
> write.amelia(obj=a.out, file.stem = "outdata", format = "dta")
```

### 5.2.2 Combining Multiple Amelia Runs

The EMB algorithm is what computer scientists call *embarrassingly parallel*, meaning that it is simple to separate each imputation into parallel processes. With Amelia it is simple to run subsets of the imputations on different machines and then combine them after the imputation for use in analysis model. This allows for a huge increase in the speed of the algorithm.

For instance, suppose that we wanted to add another ten imputated datasets to my first call to `amelia`. First, run the function to get these additional imputations,

```
> a.out.more <- amelia(freetrade, m = 10, ts = "year",
+     cs = "country", p2s = 0)
> a.out.more

Amelia output with 10 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  15
Imputation 2:  13
Imputation 3:  11
Imputation 4:  21
Imputation 5:  13
Imputation 6:  14
Imputation 7:  11
Imputation 8:  10
Imputation 9:  12
Imputation 10:  10
```

then combine this output with our original output using the `ameliabind` function,

```
> a.out.more <- ameliabind(a.out, a.out.more)
> a.out.more

Amelia output with 15 imputed datasets.
Return code:  1
Message:  Normal EM convergence

Chain Lengths:
--------------
Imputation 1:  16
Imputation 2:  14
Imputation 3:  15
Imputation 4:  21
Imputation 5:  17
Imputation 6:  15
Imputation 7:  13
Imputation 8:  11
Imputation 9:  21
Imputation 10:  13
Imputation 11:  14
Imputation 12:  11
Imputation 13:  10
Imputation 14:  12
Imputation 15:  10
```

This function binds the two outputs into the same output so that you can pass the combined imputations easily to analysis models and diagnostics. Note that `a.out.more` now has a total of 15 imputations.

A simple way to execute a parallel processing scheme with Amelia would be to run `amelia` with `m` set to 1 on $m$ different machines, save each output using the `save` function, load them all on the same R session using `load` command and then combine them using `ameliabind`. In order to do this, however, make sure to name each of the outputs a different name so that they do not overwrite each other when loading into the same R session.

### 5.2.3 Screen Output

Screen output 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 be useful 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. Having many of these two symbols in the screen output is an indication of a problematic imputation model[7].

An example of the output when `p2s` is 2 would be

```
> amelia(freetrade, m = 1, ts = "year", cs = "country",
+     p2s = 2)

amelia starting
beginning prep functions
Variables used:  tariff polity pop gdp.pc intresmi signed fiveop usheg
running bootstrap
-- Imputation 1 --
setting up EM chain indicies

 1(44) 2(35) 3(26) 4(23) 5(18) 6(15) 7(15) 8(12) 9(10)10(7)
11(5)12(2)13(0)

 saving and cleaning


Amelia output with 1 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  13
```

## 5.3 Imputation-improving Transformations

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 Amelia 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, Amelia 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 should be declared to Amelia, whereas ordinal (including dichotomous) variables often need not be, as

[7]Problems of non-invertible matrices often mean that current guess for the covariance matrix is singular. This is a sign that there may be two highly correlated variables in the model. One way to resolve is to use a ridge prior (see 5.6.1)

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 by taking a log or square root prior to feeding the data to `amelia`) these do not need to be declared, as that would result in the transformation occurring *doubly* in the imputation model. The fully imputed data sets that are returned will always be in the form of the original data that is passed to the `amelia` routine.

### 5.3.1 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 Amelia 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.

In the `freetrade` data, one such ordinal variable is `polity` which ranges from -10 (full autocracy) to 10 (full democracy). If we tabulate this variable from one of the imputed datasets,

```
> table(a.out$imputations[[3]]$polity)

                -8                 -7                 -6
                 1                 22                  4
                -5                 -4 -3.85761223879231
                 7                  3                  1
                -2                 -1                  2
                 9                  1                  7
                 3                  4                  5
                 7                 15                 26
                 6  6.36461808491675                  7
                13                  1                  5
                 8                  9
                36                 13
```

we can see that there is one imputation between -4 and -3 and one imputation between 6 and 7. Again, the interpretation of these values is rather straightforward even if they are not strictly in the coding of the original Polity data.

Often, however, analysis models require some variables to be strictly ordinal, as for example the dependent variable must be in a logistical or Poisson 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.

For our data we can simply add `polity` to the `ords` argument:

```
> a.out1 <- amelia(freetrade, m = 5, ts = "year", cs = "country",
+     ords = "polity", p2s = 0)
> table(a.out1$imputations[[3]]$polity)

-8 -7 -6 -5 -4 -2 -1  2  3  4  5  6  7  8  9
 1 22  4  8  3  9  1  7  7 15 27 13  5 36 13
```

Now, we can see that all of the imputations fall into one of the original polity categories.

### 5.3.2 Nominal

Nominal variables[8] 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 Amelia. In our `freetrade` dataset, we have `signed` which is 1 if a country signed an IMF agreement in that year and 0 if it did not. Of course, our first imputation did not limit the imputations to these two categories

```
> table(a.out1$imputations[[3]]$signed)

                  0 0.00497669216850194  0.0650017766328817
                142                    1                   1
 0.184594066476458                    1
                  1                   26
```

In order to fix this for a $p$-category multinomial variable,Amelia 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.

For our data we can simply add `signed` to the `noms` argument:

[8]Dichotomous (two category) variables are a special case of nominal variables. For these variables, the nominal and ordinal methods of transformation in Amelia agree.

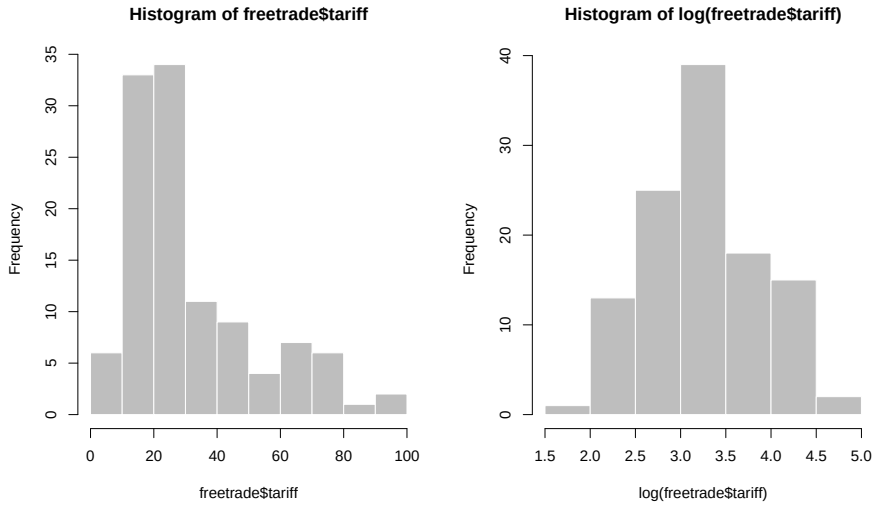


Figure 2: Histogram of `tariff` and `log(tariff)`.

```
> a.out2 <- amelia(freetrade, m = 5, ts = "year", cs = "country",
+     noms = "signed", p2s = 0)
> table(a.out2$imputations[[3]]$signed)

  0   1
144  27
```

Note that Amelia can only fit imputations into categories that exist in the original data. Thus, if there was a third category of signed, say 2, that corresponded to a different kind of IMF agreement, but it never occurred in the original data, Amelia could not match imputations to it.

Since Amelia 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, it is useful to use a ridge prior as in section 5.6.1.

### 5.3.3 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 Amelia 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.

For instance, figure 2 show the `tariff` variable clearly has positive (or, right) skew while its natural log transformation has a roughly normal distribution.

### 5.3.4 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.

### 5.3.5 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.

## 5.4 Identification Variables

Datasets often contain identification variables, such as country names, respondent numbers, or other identification 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.

If the `year` and `country` contained no information except labels, we could omit them from the imputation:

```
> amelia(freetrade, idvars = c("year", "country"))
```

Note that Amelia will return with an error if your dataset contains a factor or character variable that is not marked as a nominal or identification variable. Thus, if we were to omit the factor `country` from the `cs` or `idvars` arguments, we would receive an error:

```
> a.out2 <- amelia(freetrade, idvars = c("year"))

Amelia Error Code:  38
 The variable(s) country are "characters".  You may have wanted to set this as a
```

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.

## 5.5 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, tariffs vary 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:

$$\text{tariff}_{ti} = \beta_0 + \beta_1 t + \beta_1 t^2 + \beta_1 t^3 \ldots \tag{9}$$

And thus if we include enough higher order terms of time then the pattern between observed values of the tariff rate can be estimated. Amelia will create polynomials of time up to the user defined $k$-th order, ($k \leq 3$).

We can implement this with the `ts` and `polytime` arguments. If we thought that a second-order polynomial would help predict we could run

```
> a.out2 <- amelia(freetrade, ts = "year", cs = "country",
+     polytime = 2)
```

With this input, Amelia will add covariates to the model that correspond to time and its polynomials. These covariates will help better predict the missing values.

If cross-sectional units are specified these polynomials can be interacted with the cross-section unit 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 setting. 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 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.

The above code would predict the same trend in a variable for each country. It is clear, however, that each country will have a different time series for tariff rates, for instance. Some countries may start higher than other or possibly some countries dropped dramatically while other remained fairly constant over time. In order to capture this in the style above, we can set `intercs` to `TRUE`:

```
> a.out.time <- amelia(freetrade, ts = "year", cs = "country",
+     polytime = 2, intercs = TRUE, p2s = 2)
```

Note that attempting to use `polytime` without the `ts` argument, or `intercs` without the `cs` argument will result in an error.

Using the `tscsPlot` function (discussed below), we can see in figure 3 that we have a much better prediction about the missing values when incorporating time than when we omit it:

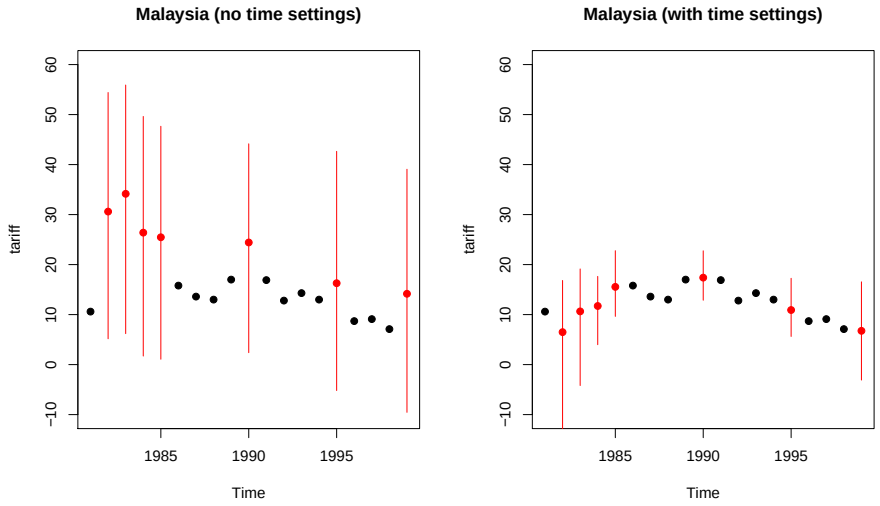


Figure 3: The increase in predictive power when using polynomials of time. The panels shows mean imputations with 95% bands (in red) and observed data point (in black). The left panel shows an imputation without using time and the right panel includes polynomials of time.

```
> tscsPlot(a.out, cs = "Malaysia", main = "Malaysia (no time settings)",
+     var = "tariff", ylim = c(-10, 60))
> tscsPlot(a.out.time, cs = "Malaysia", main = "Malaysia (with time settings)",
+     var = "tariff", ylim = c(-10, 60))
```

### 5.5.1 Lags and Leads

An alternative way of handling time-series information is to include lags and leads of certain variables into the imputation model. *Lags* are variables that take the value of another variable in the previous time period while *leads* take the value of another variable in the next time period. Many analysis models use lagged variables to deal with issues of endogeneity, thus using leads may seems strange. It is important to remember, however, that imputation models are predictive, not causal. Thus, since both past and future values of a variable are likely correlated with the present value, both lags and leads should improve the model.

If we wanted to include lags and leads of tariffs, for instance, we would simply pass this to the `lags` and `leads` arguments:

```
> a.out2 <- amelia(freetrade, ts = "year", cs = "country",
+     lags = "tariff", leads = "tariff")
```

## 5.6 Including Prior Information

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

### 5.6.1 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. This can happen when using the polynomials of time interacted with the cross section are included in the imputation model. In our running example, we used a polynomial of degree 2 and there are 9 countries. This adds $3 \times 9 = 18$ more variables to the imputation model. When these are added, the EM algorithm can become unstable, as indicated by the differing chain lengths for each imputation:

```
> a.out.time

Amelia output with 5 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  315
Imputation 2:  246
Imputation 3:  124
Imputation 4:  96
Imputation 5:  164
```

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. This can be done by including the `empri` argument. 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 and 10 percent is reasonable upper bound.

For our data, it is easy to code up a 1 percent ridge prior:

```
> a.out.time2 <- amelia(freetrade, ts = "year", cs = "country",
+     polytime = 2, intercs = TRUE, p2s = 0, empri = 0.01 *
```

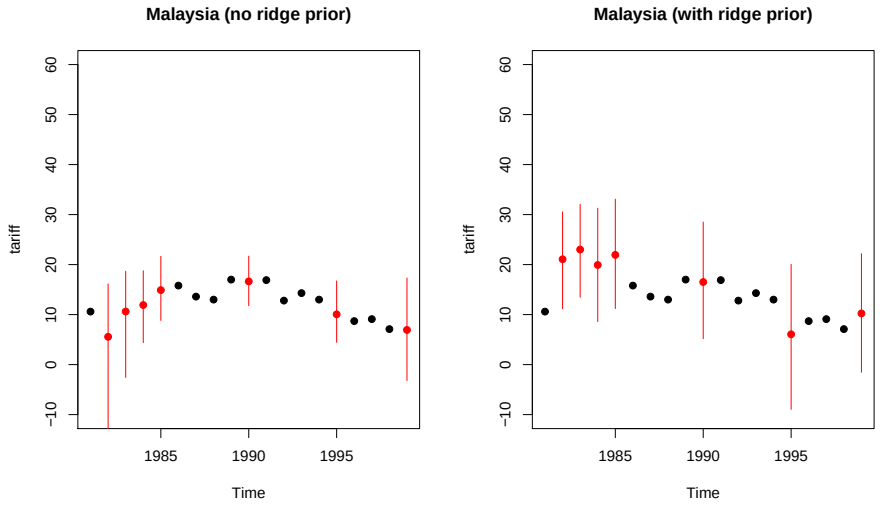


Figure 4: The difference in imputations when using no ridge prior (left) and when using a ridge prior set to 1% of the data (right).

```
+         nrow(freetrade))
> a.out.time2

Amelia output with 5 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  13
Imputation 2:  17
Imputation 3:  18
Imputation 4:  17
Imputation 5:  24
```

This new imputation model is much more stable and, as shown by using `tscsPlot`, produces about the same imputations as the original model (see figure 4):

```
> tscsPlot(a.out.time, cs = "Malaysia", main = "Malaysia (no ridge prior)",
+     var = "tariff", ylim = c(-10, 60))
> tscsPlot(a.out.time2, cs = "Malaysia", main = "Malaysia (with ridge prior)",
+     var = "tariff", ylim = c(-10, 60))
```

### 5.6.2 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 (Honaker and King, 2009).

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 1986 tariff rates in Thailand around 40%, but we have some uncertainty as to the exact value. Our prior belief about the distribution of the missing data cell, then, centers on 40 with a standard deviation that reflects the amount of uncertainty we have about our prior belief.

To input priors 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 instance, suppose that we had some expert prior information about tariff rates in Thailand. We know from the data that Thailand is missing tariff rates in many years,

```
> freetrade[freetrade$country == "Thailand", c("year",
+     "country", "tariff")]

    year  country tariff
153 1981 Thailand   32.3
154 1982 Thailand     NA
155 1983 Thailand     NA
156 1984 Thailand     NA
157 1985 Thailand   41.2
158 1986 Thailand     NA
159 1987 Thailand     NA
160 1988 Thailand     NA
161 1989 Thailand   40.8
162 1990 Thailand   39.8
163 1991 Thailand   37.8
164 1992 Thailand     NA
165 1993 Thailand   45.6
166 1994 Thailand   23.3
167 1995 Thailand   23.1
168 1996 Thailand     NA
169 1997 Thailand     NA
```

```
170 1998 Thailand   20.1
171 1999 Thailand   17.1
```

Suppose that we had expert information that tariff rates were roughly 40% in Thailand between 1986 and 1988 with about a 6% margin of error. This corresponds to a standard deviation of aobut 3. In order to include this information, we must form the priors matrix:

```
> pr <- matrix(c(158, 159, 160, 3, 3, 3, 40, 40, 40, 3,
+     3, 3), nrow = 3, ncol = 4)
> pr

     [,1] [,2] [,3] [,4]
[1,]  158    3   40    3
[2,]  159    3   40    3
[3,]  160    3   40    3
```

The first column of this matrix corresponds to the row numbers of Thailand in these three years, the second column refers to the column number of `tariff` in the data and the last two columns refer to the actual prior. Once we have this matrix, we can pass it to `amelia`,

```
> a.out.pr <- amelia(freetrade, ts = "year", cs = "country",
+     priors = pr)
```

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. We could get roughly the same prior as above by utilizing this method. Our margin of error implies that we would want imputations between 34 and 46, so our matrix would be

```
> pr.2 <- matrix(c(158, 159, 160, 3, 3, 3, 34, 34, 34,
+     46, 46, 46, 0.95, 0.95, 0.95), nrow = 3, ncol = 5)
> pr.2

     [,1] [,2] [,3] [,4] [,5]
[1,]  158    3   34   46 0.95
[2,]  159    3   34   46 0.95
[3,]  160    3   34   46 0.95
```

These priors indicate that we are 95% confident that these missing values are in the range 34 to 46.

If a prior has the value 0 in the first column, this prior will be applied to all missing values in this variable, except for explitictly set priors. Thus, we could set a prior for the entire `tariff` variable of 20, but still keep the above specific priors with the following code:

```
> pr.3 <- matrix(c(158, 159, 160, 0, 3, 3, 3, 3, 40, 40,
+     40, 20, 3, 3, 3, 5), nrow = 4, ncol = 4)
> pr.3

     [,1] [,2] [,3] [,4]
[1,]  158    3   40    3
[2,]  159    3   40    3
[3,]  160    3   40    3
[4,]    0    3   20    5
```

### 5.6.3 Logical bounds

In some cases, variables in the social sciences have known logical bounds. Proportions must be between 0 and 1 and duration data must be greater than 0, for instance. Many of these logical bounds can be handled by the proper transformation (see 5.3 for more details on the transformations handled by Amelia). In the rare case that imputations must satisfy certain logical bounds not handled by these transformations, Amelia can take draws from a truncated normal distribution in order to achieve imputations that satisfy the bounds. Note, however, that this procedure imposes extrememly strong restrictions on the imputations and can lead to lower variances than the imputation model implies. In general, building a more predictive imputation model will lead to better imputations than imposing these bounds.

Amelia implements these bounds by rejection sampling. When drawing the imputations from their posterior, we repeatedly resample until we have a draw that satisfies all of the logical constraints. You can set an upper limit on the number of times to resample with the `max.resample` arguments. Thus, if after `max.resample` draws, the imputations are still outside the bounds, Amelia will set the imputation at the edge of the bounds. Thus, if the bounds were 0 and 100 and all of the draws were negative, Amelia would simply impute 0.

As an extreme example, suppose that we know, for certain that tariff rates had to fall between 30 and 40. This, obviously, is not true, but we can generate imputations from this model. In order to specify these bounds, we need to generate a matrix of bounds to pass to the `bounds` argument. This matrix will have 3 columns: the first is the column for the bounded variable, the second is the lower bound and the third is the upper bound. Thus, to implement our bound on tariff rates (the 3rd column of the dataset), we would create the matrix,

```
> bds <- matrix(c(3, 30, 40), nrow = 1, ncol = 3)
> bds

     [,1] [,2] [,3]
[1,]    3   30   40
```

which we can pass to the `bounds` argument,

```
> a.out.bds <- amelia(freetrade, ts = "year", cs = "country",
+     bounds = bds, max.resample = 1000)
```

```
-- Imputation 1 --

 1  2  3  4  5  6  7  8  9 10 11 12 13

-- Imputation 2 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18

-- Imputation 3 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14

-- Imputation 4 --

 1  2  3  4  5  6  7  8  9 10 11 12

-- Imputation 5 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
```

The difference in results between the bounded and unbounded model are not obvious from the output, but inspection of the imputed tariff rates for Malaysia in figure 5 shows that there has been a drastic restriction of the imputations to the desired range:

```
> tscsPlot(a.out, cs = "Malaysia", main = "No logical bounds",
+     var = "tariff", ylim = c(-10, 60))
> tscsPlot(a.out.bds, cs = "Malaysia", main = "Bounded between 30 and 40",
+     var = "tariff", ylim = c(-10, 60))
```

Again, analysts should be extremely cautious when using these bounds as they can seriously affect the inferences from the imputation model, as shown in this example. Even when logical bounds exist, we recommend simply imputing variables normally, as the violation of the logical bounds represents part of the true uncertainty of imputation.

## 5.7 Diagnostics

Amelia currently provides a number of diagnostic tools to inspect the imputations that are created.

### 5.7.1 Comparing Densities

One check on the plausibility of the imputaiton model is check the distribution of imputed values to the distribution of observed values. Obviously we cannot expect, *a priori*, that these distribution will be identical as the missing values may differ systematically from the observed value–this is whole reason to impute to begin with!

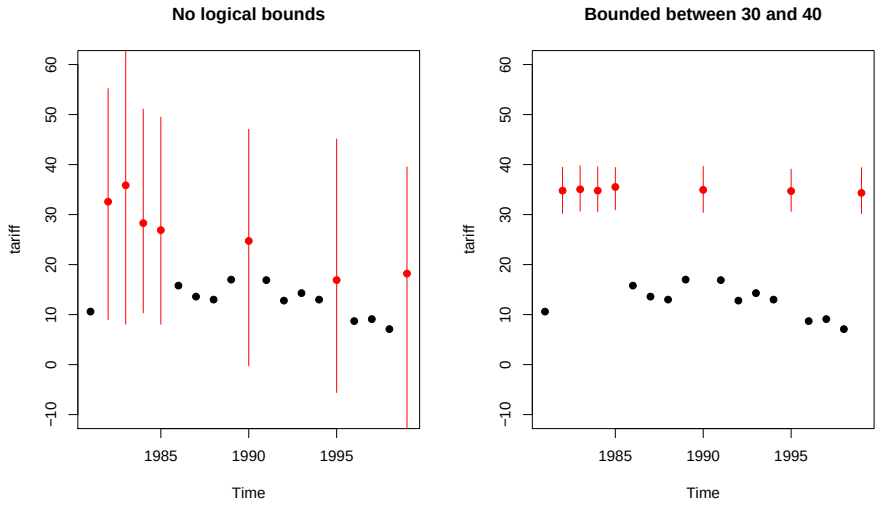


Figure 5: On the left are the original imputations without logical bounds and on the right are the imputation after imposing the bounds.

Imputations with strange distributions or those that are far from the observed data may indicate that imputation model needs at least some investigation and possibly some improvement.

The `plot` method works on output from `amelia` and, by default, shows for each variable a plot of the relative frequencies of the observed data with an overlay of the relative frequency of the imputed values.

```
> plot(a.out, which.vars = 3:6)
```

where the argument `which.vars` indicates which of the variables to plot (in this case, we are taking the 3rd through the 6th variables).

The imputed curve (in red) 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 across each of the $m$ datasets and use that value for the density plot. The black distributions are the those of the observed data. When variables are completely observed, their densities are plotted in blue. 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 `?`. 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.

We can also use the function `compare.density` directly to make these plots for an individual variable:

```
> compare.density(a.out, var = "signed")
```

### 5.7.2 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

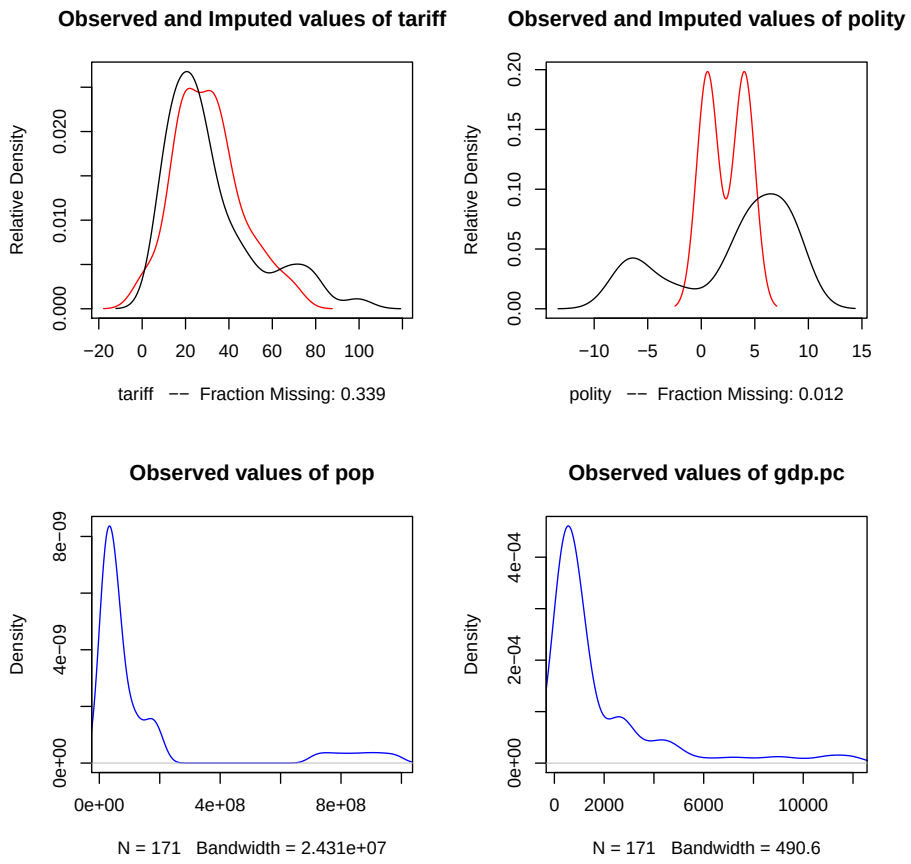


Figure 6: The output of the `plot` method as applied to output from `amelia`. In the upper panels, the distribution of mean imputations (in red) is overlayed on the distribution of observed values (in black) for each variable. In the lower panels, there are no missing values and the distribution of observed values is simply plotted (in blue). Note that how imputed tariff rates are very similar to observed tariff rates, but the imputation of the Polity score are quite different. This is plausible if different types of regimes tend to be missing at different rates.

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.

For example, we can run the overimputation diagnostic on our data by running

```
> overimpute(a.out, var = "tariff")
```

Our overimputation diagnostic, shown in 7, 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, Amelia 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 8. 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.

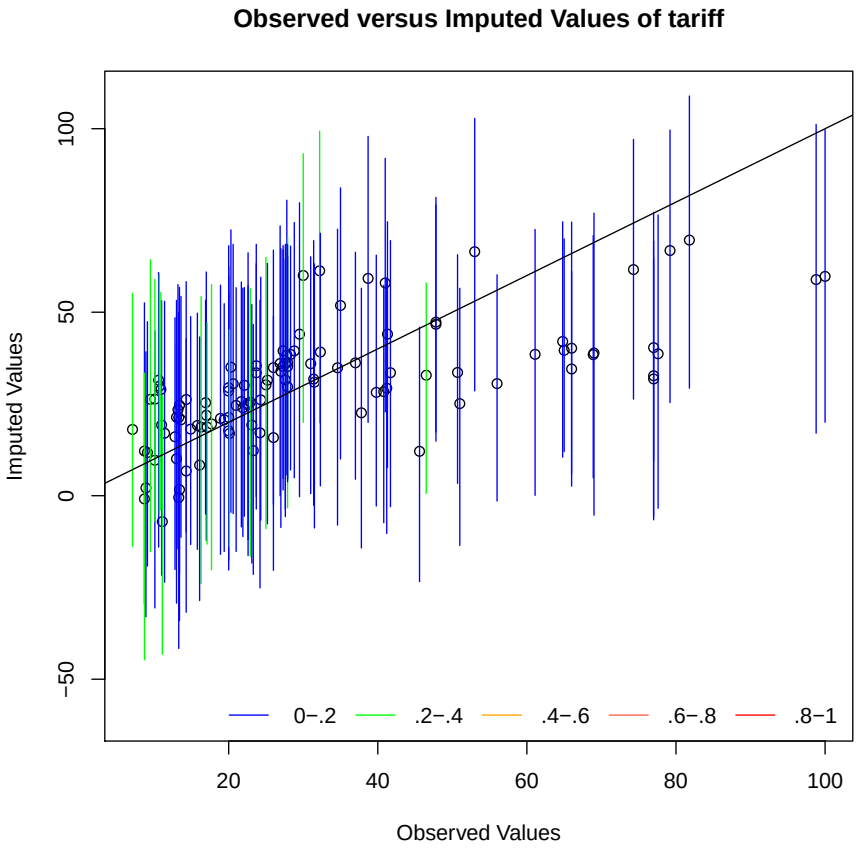


Figure 7: An example of the overimputation diagnostic graph. Here ninety percent confidence intervals are constructed that detail where an observed value would have been imputed had it been missing from the dataset, given the imputation model. The dots represent the mean imputation. Around ninety percent 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.

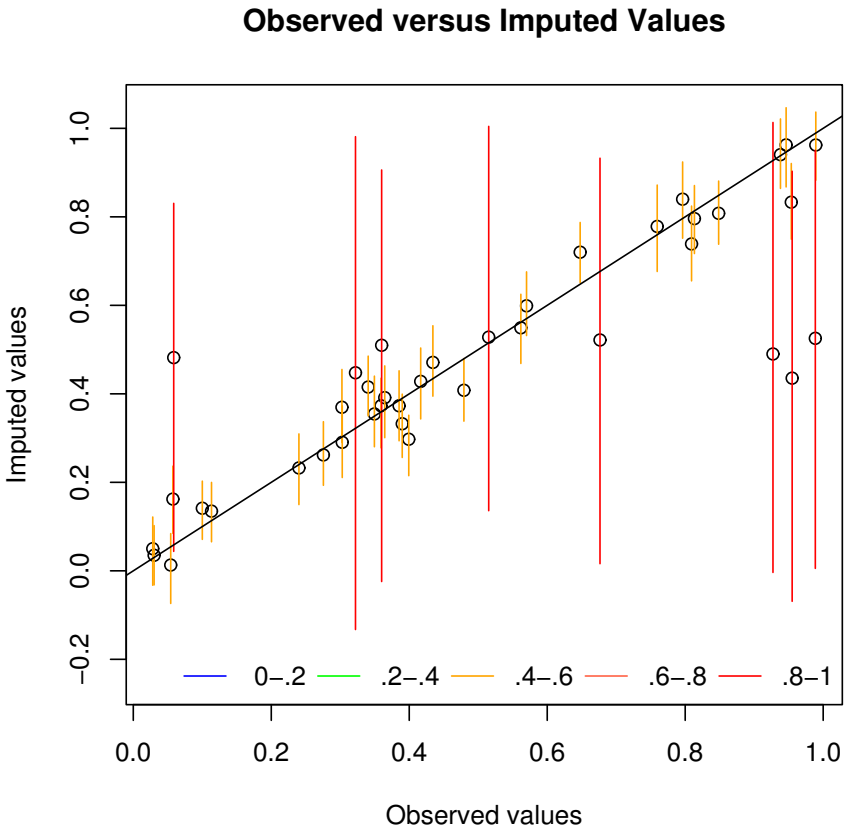


Figure 8: Another example of the overimpute diagnostic graph. Note that the red lines are those observations that have fewer covariates observed and have a higher variance in the imputation.

### 5.7.3 Overdispersed Starting Values

If the data given to Amelia 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 variable were accidentally included in the imputation model twice.

Amelia 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.

For our original model, this is a simple call to `disperse`:

```
> disperse(a.out, dims = 1, m = 5)
> disperse(a.out, dims = 2, m = 5)
```

where `m` designates the number of places to start EM chains from and `dims` are the number of dimensions of the principal components to show.

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 5.7.3 show two examples of these 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 Amelia 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.

A few of the iterations of this diagnostic can ending up in vastly different locations of the parameter space. This can happen for a variety of reasons. For instance, suppose that we created another dataset and accidently included a linear function of another variable in this dataset:

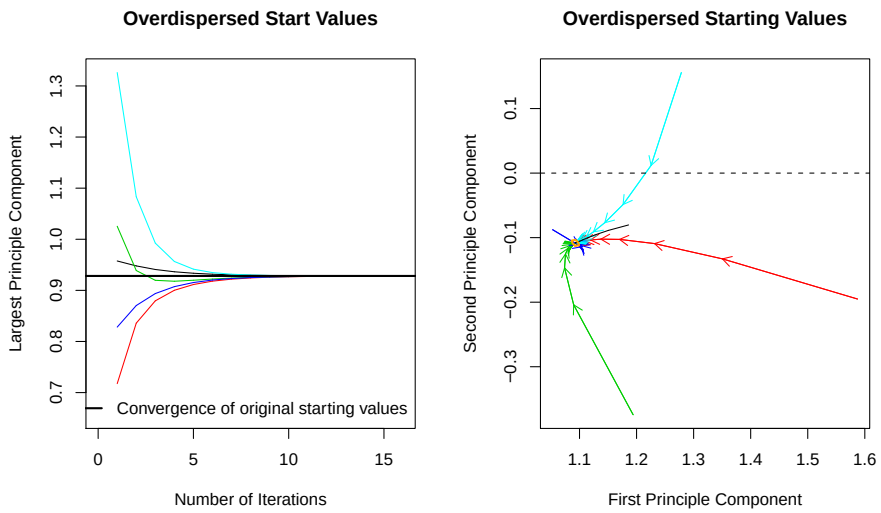


Figure 9: A plot from the overdispersion diagnostic where all EM chains are converging to the same mode, regardless of starting value. On the left, the $y$-axis represents movement in the (very high dimensional) parameter space, and the $x$-axis represents the iteration number of the chain. On the right, we visualize 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.

```
> freetrade2 <- freetrade
> freetrade2$tariff2 <- freetrade2$tariff * 2 + 3
```

If we tried to impute this dataset, Amelia could draw imputations without any problems:

```
> a.out.bad <- amelia(freetrade2, ts = "year", cs = "country")

-- Imputation 1 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

-- Imputation 2 --

 1  2  3  4  5  6  7  8  9 10 11 12

-- Imputation 3 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17

-- Imputation 4 --
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15

-- Imputation 5 --

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

> a.out.bad

Amelia output with 5 imputed datasets.
Return code:  1
Message:  Normal EM convergence.

Chain Lengths:
--------------
Imputation 1:  15
Imputation 2:  12
Imputation 3:  17
Imputation 4:  15
Imputation 5:  20
```

But if we were to run `disperse`, we would end up with the problematic figure 5.7.3:

```
> disperse(a.out.bad, dims = 1, m = 5)
```

While this is a special case of a problematic likelihood, situations very similar to this can go undetected without using the proper diagnostics. 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).

### 5.7.4 Time-series plots

As discussed above, information about time trends and fixed effects can help produce better imputations. One way to check the plausibility of our imputation model is to see how it predicts missing values in a time series. If the imputations for the Malaysian tariff rate were drastically higher in 1990 than the observed years of 1989 or 1991, we might worry that there is a problem in our imputation model. Checking these time series is easy to do with the `tscsPlot` command. Simply choose the variable (with the `var` argument) and the cross-section (with the `cs` argument) to plot the observed time-series along with distributions of the imputed values for each missing time period. For instance, we can run

```
> tscsPlot(a.out.time, cs = "Malaysia", main = "Malaysia (with time settings)",
+     var = "tariff", ylim = c(-10, 60))
```

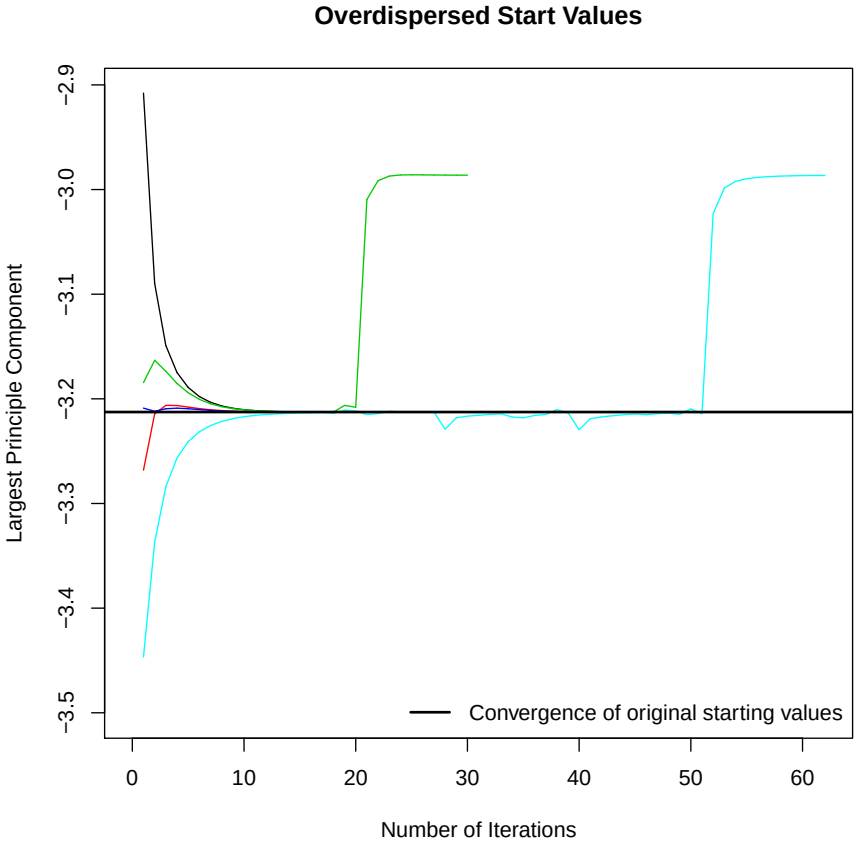


Figure 10: A problematic plot from the overdispersion diagnostic showing that EM chains are converging to one of two different modes, depending upon the starting value of the chain.

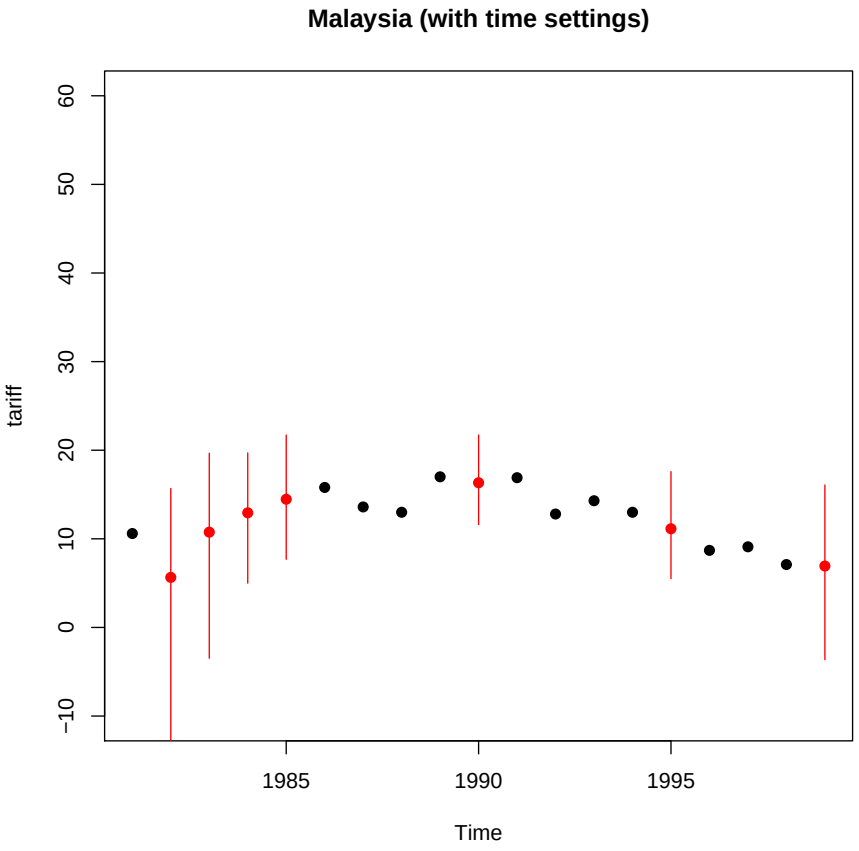


Figure 11: Tariff rates in Malaysia, 1980-2000. An example of the `tscsPlot` function, the black points are observed values of the time series and the red points are the mean of the imputation distributions. The red lines represent the 95% confidence bands of the imputation distribution.

to get the plot in figure 5.7.4. Here, the black point are observed tariff rates for Malaysia from 1980 to 2000. The red points are the mean imputation for each of the missing values, along with their 95% confidence bands. We draw these bands by imputing each of missing values 100 times to get the imputation distribution for that observation.

In figure 5.7.4, we can see that the imputed 1990 tariff rate is quite in line with the values around it. Notice also that values toward the beginning and end of the time series have higher imputation variance. This occurs because the fit of the polynomials of time in the imputation model have higher variance at the beginning and end of the time series. This is intuitive because these points have fewer neighbors from which to draw predictive power.

A word of caution is in order. As with comparing the histograms of imputed and obseved values, there could be reasons that the missing values are systematically different than the observed time series. For instance, if there had been a major financial crisis in Malaysia in 1990 which caused the government to close off trade, then we would expect that the missing tariff rates should be quite different than the observed time series. If we have this information in our imputation model, we might expect to see out-of-line imputations in these time-series plots. If, on the other hand, we did not have this information, we might see "good" time-series plots that fail to point out this violation of the MAR assumption. Our imputation model would produce poor estimates of the missing values since it would be unaware that both the missingness and the true unobserved tariff rate depend on another variable. Hence, the `tscsPlot` is useful for finding obvious problems in imputation model and comparing the efficiency of various imputation models, but it cannot speak to the untestable assumption of MAR.

### 5.7.5 Missingness maps

One useful tool for exploring the missingness in a dataset is a *missingness map*. This is a map that visualizes the dataset a grid and colors the grid by missingness status. The column of the grid are the variables and the rows are the observations, as in any spreadsheet program. This tool allows for a quick summary of the patterns of missingness in the data.

If we simply call the `missmap` function on our output from `amelia`,

```
> missmap(a.out)
```

we get the plot in figure 5.7.5. The `missmap` function arrange the columns so that the variables are in decreasing order of missingness from left to right. If the `cs` argument was set in the `amelia` function, the labels for the rows will indicate where each of the cross-sections begin.

In figure 5.7.5, it is clear that the tariff rate is the variable most missing in the data and it tends to be missing in blocks of a few observations. Gross international reserves (`intresmi`) and financial openness (`fivop`), on the other hand, are missing mostly at the end of each cross-section. This suggests *missingness by merging*, when variables with different temporal coverages are merged to make one dataset. Sometimes this kind of missingness is an artifact of the date at which the data was

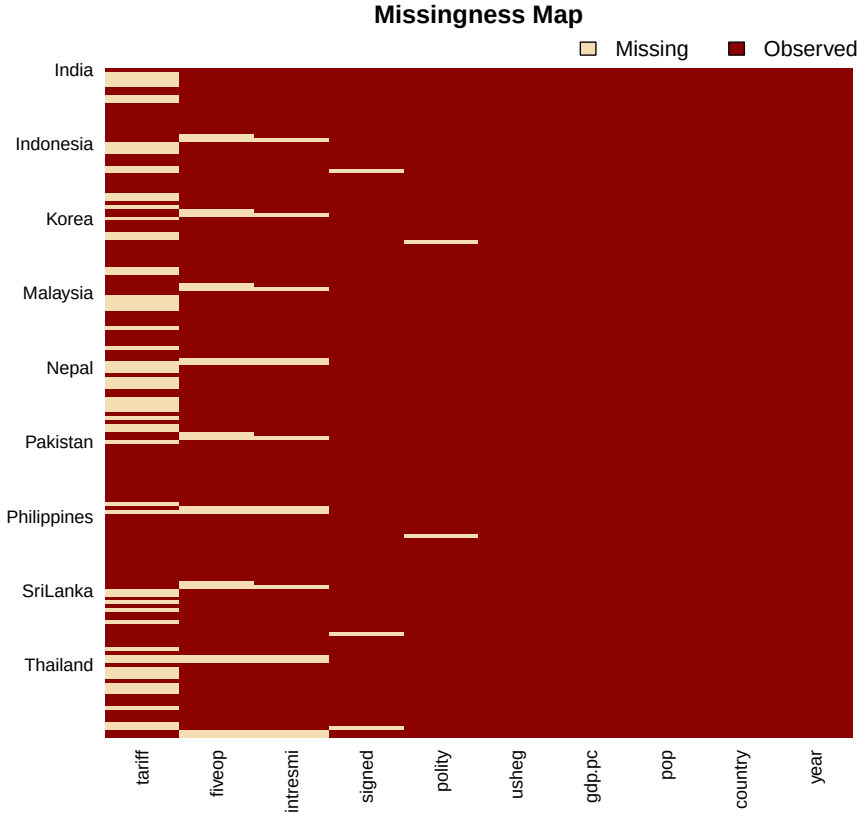


Figure 12: Missingness map of the `freetrade` data. Missing values are in tan and observed values are in red.

merged and researchers can resolve it by finding updated versions of the relevant variables.

The missingness map is an important tool for understanding the patterns of missingness in the data and can often indicate potential ways to improve the imputation model or data collection process.

## 5.8 Analysis Models

Imputation is most often a data processing step as opposed to a final model in of itself. To this end, it is easy to to pass output from `amelia` to other functions. The easiest and most integrated way to run an analysis model is to pass the output to the `zelig` function from the `Zelig` package. For example, in Milner and Kubota (2005), the dependent variable was tariff rates. We can replicate table 5.1 from their analysis with the original data simply by running

```
> require(Zelig)
> z.out <- zelig(tariff ~ polity + pop + gdp.pc + year +
+     country, data = freetrade, model = "ls")

> summary(z.out)

Call:
zelig(formula = tariff ~ polity + pop + gdp.pc + year + country,
    model = "ls", data = freetrade)

Residuals:
     Min       1Q   Median       3Q      Max
-30.7640  -3.2595   0.0868   2.5983  18.3097

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         1.97e+03   4.02e+02    4.91  3.6e-06
polity             -1.37e-01   1.82e-01   -0.75     0.45
pop                -2.02e-07   2.54e-08   -7.95  3.2e-12
gdp.pc              6.10e-04   7.44e-04    0.82     0.41
year               -8.71e-01   2.08e-01   -4.18  6.4e-05
countryIndonesia   -1.82e+02   1.86e+01   -9.82  3.0e-16
countryKorea       -2.20e+02   2.08e+01  -10.61  < 2e-16
countryMalaysia    -2.25e+02   2.17e+01  -10.34  < 2e-16
countryNepal       -2.16e+02   2.25e+01   -9.63  7.7e-16
countryPakistan    -1.55e+02   1.98e+01   -7.84  5.6e-12
countryPhilippines -2.04e+02   2.09e+01   -9.77  3.7e-16
countrySriLanka    -2.09e+02   2.21e+01   -9.46  1.8e-15
countryThailand    -1.96e+02   2.10e+01   -9.36  3.0e-15

Residual standard error: 6.22 on 98 degrees of freedom
Multiple R-squared: 0.925,        Adjusted R-squared: 0.915
F-statistic:  100 on 12 and 98 DF,  p-value: <2e-16
```

Running the same model with imputed data is almost identical. Simply replace the original data set with the imputations from the `amelia` output:

```
> z.out.imp <- zelig(tariff ~ polity + pop + gdp.pc + year +
+     country, data = a.out$imputations, model = "ls")

How to cite this model in Zelig:
Kosuke Imai, Gary King, and Oliva Lau. 2007. "ls: Least Squares Regression for Co

> summary(z.out.imp)

  Model: ls
  Number of multiply imputed data sets: 5

Combined results:

Call:
zelig(formula = tariff ~ polity + pop + gdp.pc + year + country,
    model = "ls", data = a.out$imputations)

Coefficients:
                       Value Std. Error  t-stat  p-value
(Intercept)         2.392e+03  7.394e+02  3.2347 0.004913
polity              4.475e-02  3.681e-01  0.1216 0.904072
pop                -8.198e-08  5.643e-08 -1.4528 0.171140
gdp.pc              7.330e-05  1.625e-03  0.0451 0.964523
year               -1.137e+00  3.804e-01 -2.9898 0.008001
countryIndonesia   -8.541e+01  4.117e+01 -2.0746 0.061959
countryKorea       -1.092e+02  4.399e+01 -2.4833 0.026662
countryMalaysia    -1.104e+02  4.892e+01 -2.2565 0.045645
countryNepal       -1.081e+02  4.769e+01 -2.2663 0.041917
countryPakistan    -6.123e+01  4.453e+01 -1.3749 0.196586
countryPhilippines -9.968e+01  4.665e+01 -2.1369 0.055662
countrySriLanka    -9.677e+01  5.009e+01 -1.9320 0.080360
countryThailand    -9.499e+01  4.520e+01 -2.1018 0.057144

For combined results from datasets i to j, use summary(x, subset = i:j).
For separate results, use print(summary(x), subset = i:j).
```

Zelig is one way to run analysis models on imputed data, but certainly not the only way. The `imputations` list in the `amelia` output contains each of the imputed datasets. Thus, users could simply program a loop over the number of imputations and run the analysis model on each imputed dataset and combine the results using the rules described in King et al. (2001) and Schafer (1997). Furthermore, users can easily export their imputations using the `write.amelia` function as described in 5.2.1 and use statistical packages other than R for the analysis model.

## 5.9 The `amelia` class

The output from the `amelia` function is an instance of the S3 class "amelia." Instances of the `amelia` class contain much more than simply the imputed datasets. The `mu` object of the class contains the posterior draws of the means of the complete data. The `covMatrices` contains the posterior draws of the covariance matrices of the complete data. Note that these correspond to the variables as they are sent to the EM algorithm. Namely, they refer to the variables after being transformed, centered and scaled.

The `iterHist` object is a list of `m` 3-column matrices. Each row of the matrices corresponds to an iteration of the EM algorithm. The first column indicates how many parameters had yet to converge at that iteration. The second column indicates if the EM algorithm made a step that decreased the number of converged parameters. The third column indicates whether the covariance matrix at this iteration was singular. Clearly, the last two columns are meant to indicate when the EM algorithm enters a problematic part of the parameter space.

# 6 AmeliaView Menu Guide

Below is a guide to the AmeliaView menus with references back to the users's guide. The same principles from the user's guide apply to AmeliaView. The only difference is how you interact with the program. Whether you use the GUI or the command line versions, the same underlying code is being called, and so you can read the command line-oriented versions of this manual even if you intend to use the GUI.

## 6.1 Loading AmeliaView

The easiest way to load AmeliaView is to open an R session and type the following two commands:

```
> library(Amelia)
> AmeliaView()
```

This will bring up the AmeliaView window on any platform.

On the Windows operating system, there is an alternative way to start AmeliaView from the Desktop. See section 4.1 for a guide on how to install this version. Once installed, there should be a Desktop icon for AmeliaView. Simply double-click this icon and the AmeliaView window should appear. If, for some reason, this approach does not work, simply open an R session and use the approach above.

## 6.2 Step 1 - Input

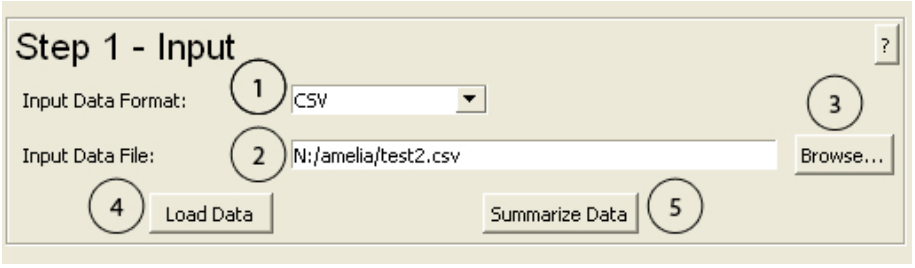


Figure 13: Detail for step 1 on the front page of AmeliaView.

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). Note that when using a CSV file, Amelia assumes that your file has a header (that is, a row at the top of the data indicating the variable names).

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. **Browse** - Find files on the system.

4. **Load Data** - Loads the data in the "Input Data File" line. Once the file is loaded, you can reload a different file, but you will lose any work on currently loaded file.

5. **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.

## 6.3 Step 2 - Options

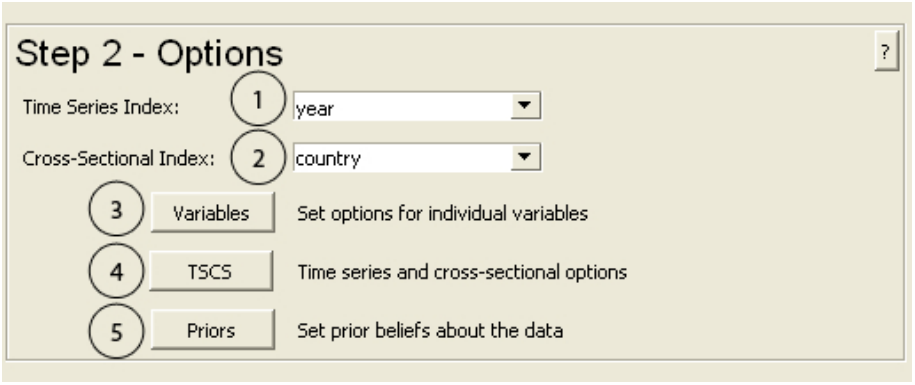


Figure 14: Detail for step 2 on the front page of AmeliaView.

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 6.3.1 for more information.

4. **TSCS** - Becomes available after you set the Time Series variable. See 6.3.2 for more information.

5. **Priors** - Becomes available after you load the data. See 6.3.3 for more information.

### 6.3.1 Variables Dialog

1. **Variable Transformations** - Choose the transformation that best tailors the variable to the multivariate normal, if appropriate. See 5.3 on Transformations

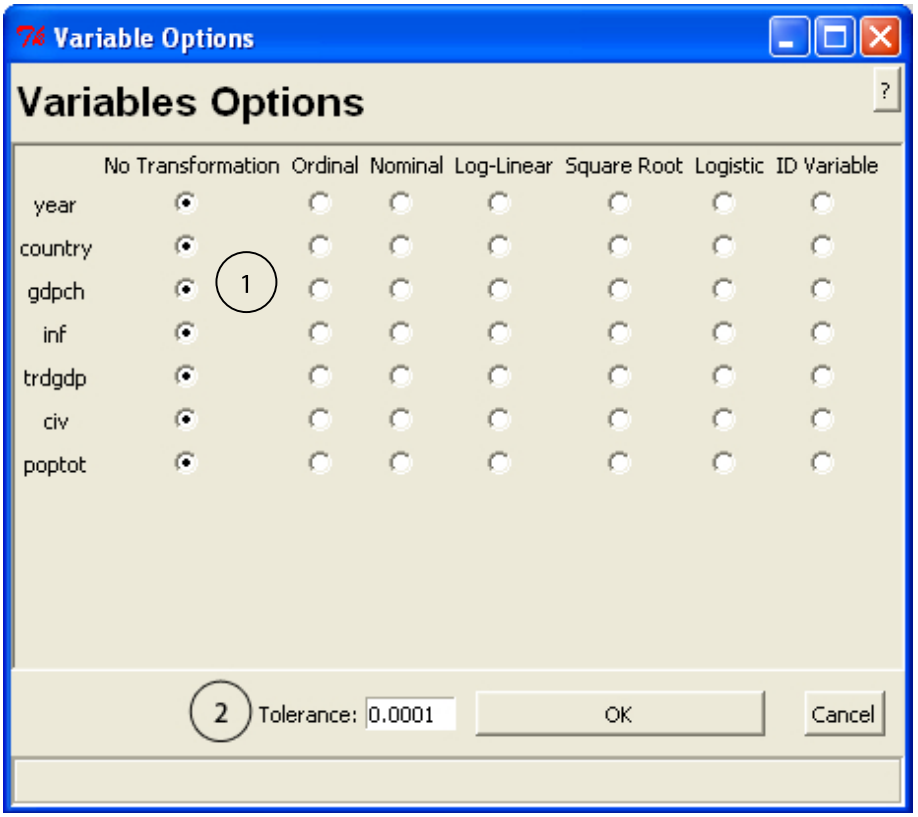


Figure 15: Detail for Variable Options dialog.

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.

### 6.3.2 Time Series Cross Sectional 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 5.5 above.

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

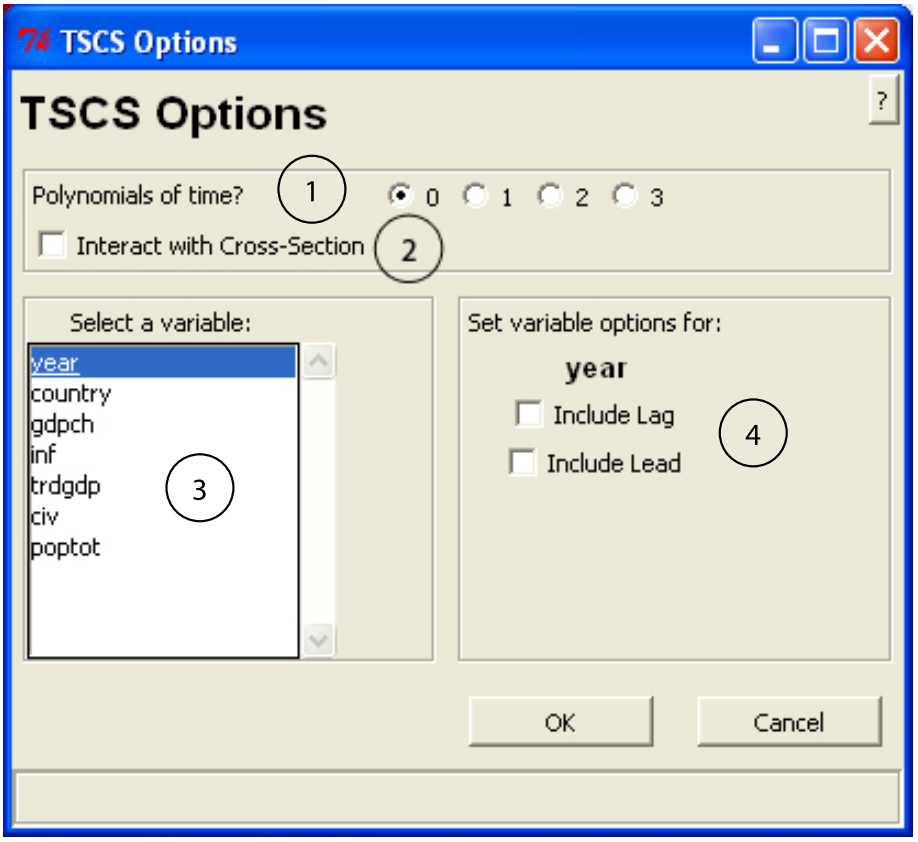


Figure 16: Detail for Time-Series-Cross-Section Options dialog.

4. **Lag Settings** - Choose to include lags and leads in the data set to handle the effects of time. See 5.5.1 above.

### 6.3.3 Priors 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 (see 5.6.1).

2. **Set Observational Priors** - Set prior beliefs about ranges for individual missing observations. For more information about observational priors, see 5.6.2.

### 6.3.4 Observational Priors

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.

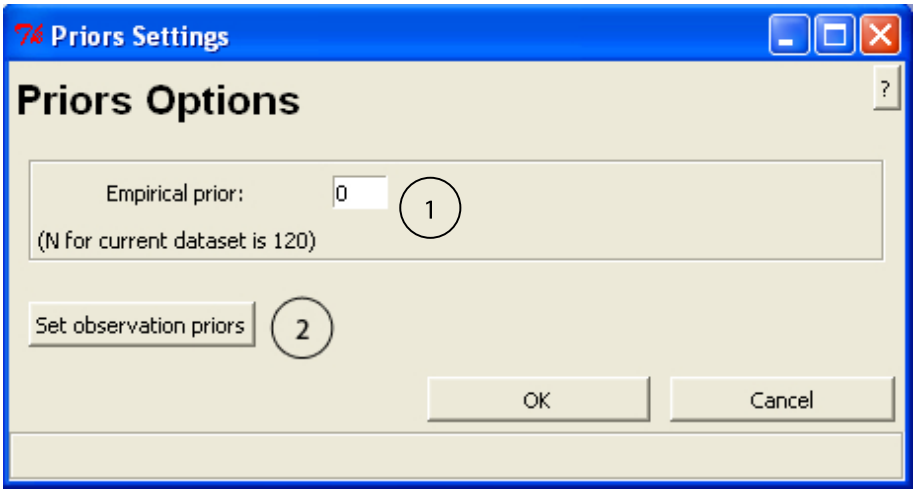


Figure 17: Detail for Priors Options dialog.

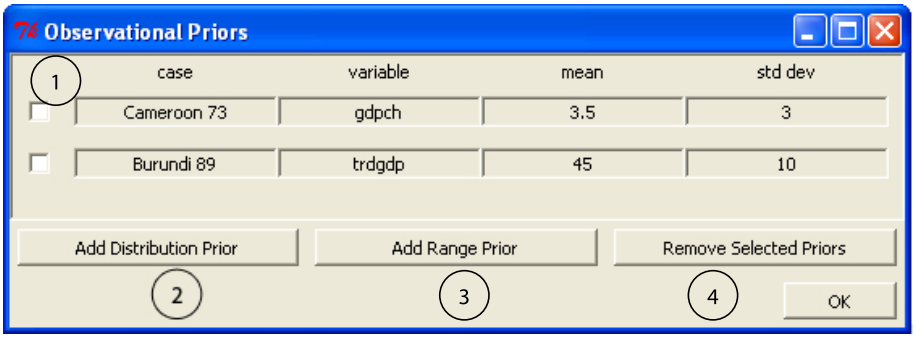


Figure 18: Detail for Observational Priors dialog

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

### 6.3.5 Add Distribution 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.

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.

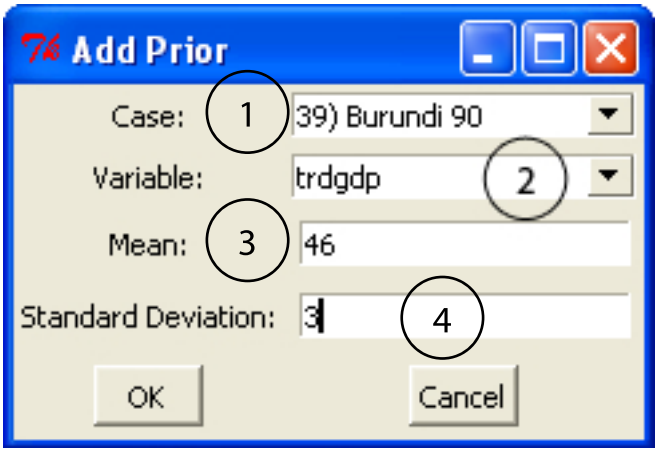


Figure 19: Detail for Add Distributional Prior dialog

### 6.3.6 Add Range Prior

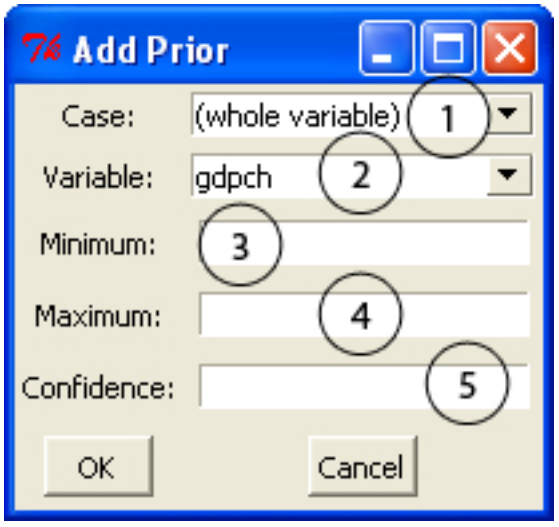


Figure 20: Detail for Add Range Prior dialog

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. **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.

## 6.4 Step 3 - Output

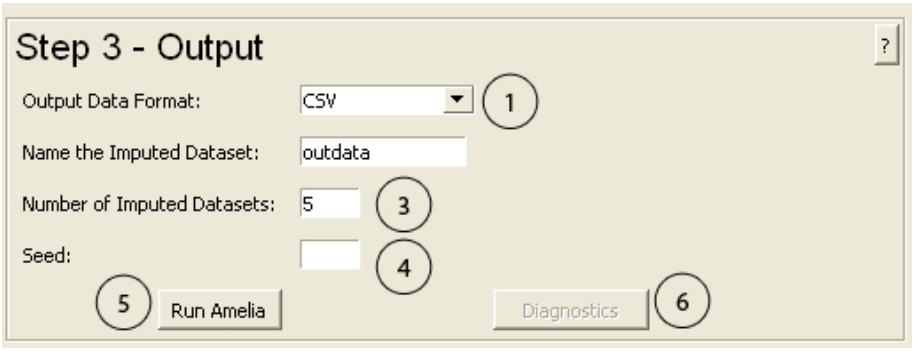


Figure 21: Detail for step 3 on the front page of AmeliaView.

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), Stata (.DTA), R save object (.RData), or to hold it in R memory. This last option will only work if you have called AmeliaView from an R session and want to return to the R command line to work with the output. It will have the name in memory from "Name of Imputed Datasets".

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. **Seed** - Sets the seed for the random number generator used by Amelia. Useful if you need to have the same output twice.

5. **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.

6. **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.

### 6.4.1 Diagnostics Dialog

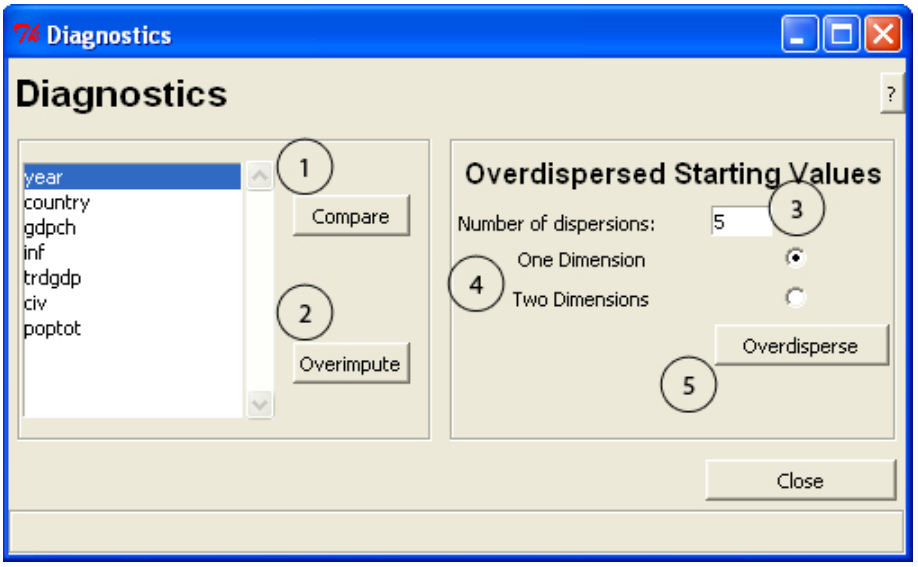


Figure 22: Detail for Diagnostics dialog.

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 Amelia algorithm from multiple start values that are drawn randomly.

# 7 Reference to Amelia's Functions

## 7.1 africa: Economic and Political Indictors in 6 African States

### Description

Data on a few economic and political variables in six African States from 1972-1991. The variables are year, country name, Gross Domestic Product per capita, inflation, trade as a percentage of GDP, a measure of civil liberties and total population. The data is from the Africa Research Program. A few cells are missing.

### Usage

```
africa
```

### Format

A data frame with 7 variables and 120 observations.

### Source

Africa Research Program `http://africa.gov.harvard.edu`

## 7.2 amelia: AMELIA: Mutliple Imputation of Incomplete Multivariate Data

### Description

Runs the bootstrap EM algorithm on incomplete data and creates imputed datasets.

### Usage

```
## Default S3 method:
amelia(x, m = 5, p2s = 1,frontend = FALSE, idvars = NULL,
       ts = NULL, cs = NULL, polytime = NULL, intercs = FALSE,
       lags = NULL, leads = NULL, startvals = 0, tolerance = 0.0001,
       logs = NULL, sqrts = NULL, lgstc = NULL, noms = NULL, ords = NULL,
       incheck = TRUE, collect = FALSE, arglist = NULL, empri = NULL,
       priors = NULL, autopri = 0.05, emburn = c(0,0), bounds = NULL,
       max.resample = 100, ...)

## S3 method for class 'amelia':
amelia(x, m = 5, p2s = 1, frontend = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | either a matrix, data.frame or a object of class "amelia". The first two will call the default S3 method. The third a convenient way to perform more imputations with the same parameters. |
| 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.

`arglist` an object of class "ameliaArgs" from a previous run of Amelia. Including this object will use the arguments from that run.

`empri` number indicating level of the empirical (or ridge) prior. This prior shrinks 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, perhaps 0.5 to 1 percent of the rows of the data; a reasonable upper bound is around 10 percent of the rows of the data.

`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.

`emburn` a numeric vector of length 2, where `emburn[1]` is a the minimum EM chain length and `emburn[2]` is the maximum EM chain length. These are ignored if they are less than 1.

`bounds` a three column matrix to hold logical bounds on the imputations. Each row of the matrix should be of the form `c(column.number, lower.bound,upper.bound)` See Details below.

`max.resample` an integer that specifies how many times Amelia should redraw the imputed values when trying to meet the logical constraints of `bounds`. After this value, imputed values are set to the bounds.

`...` further arguments to be passed.

## 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 datatsets with no missing values. The algorithm 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 produce the `m` complete datasets where the observed values are the same and the unobserved values are drawn from their posterior distributions.

The function will start a "fresh" run of the algorithm if `x` is either a incomplete matrix or data.frame. In this method, all of the options will be user-defined or set to their default. If `x` the output of a previous Amelia run (that is, an object of class "amelia"), then Amelia will run with the options used in that previous run. This is a convenient way to run more imputations of the same model.

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.

In addition to priors, Amelia allows for logical bounds on variables. The `bounds` argument should be a matrix with 3 columns, with each row referring to a logical bound on a variable. The first column should be the column number of the variable to be bounded, the second column should be the lower bounds for that variable, and the third column should be the upper bound for that variable. As Amelia enacts these bounds by resampling, particularly poor bounds will end up resampling forever. Amelia will stop resampling after `max.resample` attempts and simply set the imputation to the relevant bound.

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.

## Value

An instance of S3 class "amelia" with the following objects:

| | |
|---|---|
| `imputations` | a list of length `m` with an imputed dataset in each entry. The class (matrix or data.frame) of these entries will match `x`. |
| `m` | an integer indicating the number of imputations run. |
| `missMatrix` | a matrix identical in size to the original dataset with 1 indicating a missing observation and a 0 indicating an observed observation. |
| `theta` | An array with dimensions $(p+1)$ by $(p+1)$ by $m$ (where $p$ is the number of variables in the imputations model) holding the converged parameters for each of the `m` EM chains. |
| `mu` | A $p$ by $m$ matrix of of the posterior modes for the complete-data means in each of the EM chains. |
| `covMatrices` | An array with dimensions $(p)$ by $(p)$ by $m$ where the first two dimensions hold the posterior modes of the covariance matrix of the complete data for each of the EM chains. |
| `code` | a integer indicating the exit code of the Amelia run. |

| | |
|---|---|
| `message` | an exit message for the Amelia run |
| `iterHist` | a list of iteration histories for each EM chain. See documentation for details. |
| `arguments` | a instance of the class "ameliaArgs" which holds the arguments used in the Amelia run. |

Note that the `theta`, `mu` and `covMatrcies` objects refers to the data as seen by the EM algorithm and is thusly centered, scaled, stacked, tranformed and rearranged. See the manual for details and how to access this information.

## Author(s)

James Honaker, Gary King, Matt Blackwell

## See Also

For imputation diagnostics, `missmap`, `compare.density`, `overimpute` and `disperse`. For time series plots, `tscsPlot`. Also: `plot.amelia`, `write.amelia`, and `ameliabind`.

## Examples

```
data(africa)
a.out <- amelia(x = africa, cs = "country", ts = "year", logs = "gdp_pc")
summary(a.out)
plot(a.out)
```

## 7.3 `compare.density`: Compare observed versus imputed densities

### Description

Plots smoothed density plots of observed and imputed values from output from the `amelia` function.

### Usage

```
compare.density(output, var, col = c("red","black"), scaled = FALSE,
 lwd = 1, main, xlab, ylab, legend = TRUE, frontend = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `output` | output from the function `amelia`. |
| `var` | column number or variable name of the variable to plot. |
| `col` | a vector of length 2 containing the color to plot the (1) imputed density and (2) the observed density. |
| `scaled` | a logical indicating if the two densities should be scaled to reflect the difference in number of units in each. |
| `lwd` | the line width of the density plots. |
| `main` | main title of the plot. The default is to title the plot using the variable name. |
| `xlab` | the label for the x-axis. The default is the name of the variable. |
| `ylab` | the label for the y-axis. The default is "Relative Density." |
| `legend` | a logical value indicating if a legend should be plotted. |
| `frontend` | a logical value used internally for the Amelia GUI. |
| `...` | further graphical parameters for the plot. |

### Details

This function first plots a density plot of the observed units for the variable `var` in `col[2]`. The the function plots a density plot of the mean or modal imputations for the missing units in `col[1]`. If a variable is marked "ordinal" or "nominal" with the `ords` or `noms` options in `amelia`, then the modal imputation will be used. If `legend` is `TRUE`, then a legend is plotted as well.

### References

Abayomi, K. and Gelman, A. and Levy, M. 2005 "Diagnostics for Multivariate Imputations," *Applied Statistics*. 57,3: 273–291.

### See Also

For more information on how densities are computed, `density`; Other imputation diagnostics are `overimpute`, `disperse`, and `tscsPlot`.

## 7.4 `disperse`: Overdispersed starting values diagnostic for multiple imputation

### Description

A visual diagnostic of EM convergence from multiple overdispersed starting values for an output from `amelia`.

### Usage

```
disperse(output, m = 5, dims = 1, p2s = 0, frontend = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `output` | output from the function `amelia`. |
| `m` | the number of EM chains to run from overdispersed starting values. |
| `dims` | the number of principle components of the parameters to display and assess convergence on (up to 2). |
| `p2s` | an integer that controls printing to screen. 0 (default) indicates no printing, 1 indicates normal screen output and 2 indicates diagnostic output. |
| `frontend` | a logical value used internally for the Amelia GUI. |
| `...` | further graphical parameters for the plot. |

### Details

This function tracks the convergence of `m` EM chains which start from various overdispersed starting values. This plot should give some indication of the sensitivity of the EM algorithm to the choice of starting values in the imputation model in `output`. If all of the lines converge to the same point, then we can be confident that starting values are not affecting the EM algorithm.

As the parameter space of the imputation model is of a high-dimension, this plot tracks how the first (and second if `dims` is 2) principle component(s) change over the iterations of the EM algorithm. Thus, the plot is a lower dimensional summary of the convergence and is subject to all the drawbacks inherent in said summaries.

For `dims==1`, the function plots a horizontal line at the position where the first EM chain converges. Thus, we are checking that the other chains converge close to that horizontal line. For `dims==2`, the function draws a convex hull around the point of convergence for the first EM chain. The hull is scaled to be within the tolerance of the EM algorithm. Thus, we should check that the other chains end up in this hull.

## See Also

Other imputation diagnostics are `compare.density`, `disperse`, and `tscsPlot`.

## 7.5 `freetrade`: Trade Policy and Democracy in 9 Asian States

### Description

Economic and political data on nine developing countries in Asia from 1980 to 1999. This dataset includes 9 variables including year, country, average tariff rates, Polity IV score, total population, gross domestic product per capita, gross international reserves, a dummy variable for if the country had signed an IMF agreement in that year, a measure of financial openness, and a measure of US hegemony. These data were used in Milner and Kubota (2005).

### Usage

```
freetrade
```

### Format

A data frame with 10 variables and 171 observations.

### Source

World Bank, World Trade Organization, Polity IV and others.

### References

Helen Milner and Keiko Kubota (2005), "Why the move to free trade? Democracy and trade policy in the developing countries." *International Organization*, Vol 59, Issue 1.

## 7.6 missmap: Missingness Map

### Description

Plots a missingness map showing where missingness occurs in the dataset passed to amelia.

### Usage

```
missmap(obj, legend = TRUE, col = c("wheat","darkred"), main,
        y.cex = 0.8, x.cex = 0.8, y.labels, y.at, csvar = NULL, tsvar = NULL,
```

### Arguments

| | |
|---|---|
| obj | an object of class "amelia"; typically output from the function amelia, a matrix or a dataframe. |
| legend | should a legend be drawn? |
| col | a vector of length two where the first element specifies the color for missing cells and the second element specifies the color for observed cells. |
| main | main title of the plot. Defaults to "Missingness Map". |
| x.cex | expansion for the variables names on the x-axis. |
| y.cex | expansion for the unit names on the y-axis. |
| y.labels | a vector of row labels to print on the y-axis |
| y.at | a vector of the same length as y.labels with row nmumbers associated with the labels. |
| csvar | column number or name of the variable corresponding to the unit indicator. Only used when the obj is not of class amelia. |
| tsvar | column number or name of the variable corresponding to the time indicator. Only used when the obj is not of class amelia. |
| ... | further graphical arguments. |

### Details

missmap draws a map of the missingness in a dataset using the image function. The columns are reordered to put the most missing variable farthest to the left. The rows are reordered to a unit-period order if the ts and cs arguments were passed to amelia. If not, the rows are not reordered.

The y.labels and y.at commands can be used to associate labels with rows in the data to identify them in the plot. The y-axis is internally inverted so that the first row of the data is associated with the top-most row of the missingness map. The values of y.at should refer to the rows of the data, not to any point on the plotting region.

## See Also

compare.density, overimpute, tscsPlot, image, heatmap

## 7.7 `overimpute`: Overimputation diagnostic plot

### Description

Treats each observed value as missing and imputes from the imputation model from `amelia` output.

### Usage

```
overimpute(output, var, legend = TRUE, xlab, ylab, main,
           frontend = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `output` | output from the function `amelia`. |
| `var` | column number or variable name of the variable to overimpute. |
| `legend` | a logical value indicating if a legend should be plotted. |
| `xlab` | the label for the x-axis. The default is "Observed Values." |
| `ylab` | the label for the y-axis. The default is "Imputed Values." |
| `main` | main title of the plot. The default is to smartly title the plot using the variable name. |
| `frontend` | a logical value used internally for the Amelia GUI. |
| `...` | further graphical parameters for the plot. |

### Details

This function temporarily treats each observed value in `var` as missing and imputes that value based on the imputation model of `output`. The dots are the mean imputation and the vertical lines are the 90% percent confidence intervals for imputations of each observed value. The diagonal line is the $y = x$ line. If all of the imputations were perfect, then our points would all fall on the line. A good imputation model would have about 90% of the confidence intervals containing the truth; that is, about 90% of the vertical lines should cross the diagonal.

The color of the vertical lines displays the fraction of missing observations in the pattern of missingness for that observation. The legend codes this information. Obviously, the imputations will be much tighter if there are more observed covariates to use to impute that observation.

### See Also

Other imputation diagnostics are `compare.density`, `disperse`, and `tscsPlot`.

## 7.8 plot.amelia: Summary plots for Amelia objects

### Description

Plots diagnostic plots for the output from the amelia function.

### Usage

```
## S3 method for class 'amelia':
plot(x, which.vars, compare = TRUE, overimpute =
                      FALSE, ask = par("ask"), ...)
```

### Arguments

| | |
|---|---|
| x | an object of class "amelia"; typically output from the function amelia. |
| which.vars | a vector indicating the variables to plot. The default is to plot all of the numeric variables that were actually imputed. |
| compare | plot the density comparisons for each variable? |
| overimpute | plot the overimputation for each variable? |
| ask | prompt user before changing pages of a plot? |
| ... | further graphical arguments. |

### See Also

compare.density, overimpute

## 7.9 summary.amelia: Summary of an Amelia object

### Description

Returns summary information from the Amelia run along with missingles information.

### Usage

```
## S3 method for class 'amelia':
summary(object, ...)
```

### Arguments

| | |
|---|---|
| object | an object of class amelia. Typically, an output from the function amelia. |
| ... | further arguments. |

### See Also

amelia, plot.amelia

## 7.10 tscsPlot: Plot observed and imputed time-series for a single cross-section

### Description

Plots a time series for a given variable in a given cross-section and provides confidence intervals for the imputed values.

### Usage

```
tscsPlot(output, var, cs, draws = 100, conf = .90,
         misscol = "red", obscol = "black", xlab, ylab, main,
         pch, ylim, xlim, ...)
```

### Arguments

| | |
|---|---|
| output | output from the function amelia. |
| var | the column number or variable name of the variable to plot. |
| cs | the name of the cross-section to plot. |
| draws | the number of imputations on which to base the confidence intervals. |
| conf | the confidence level of the confidence intervals to plot for the imputated values. |
| misscol | the color of the imputed values and their confidence intervals. |
| obscol | the color of the points for observed units. |
| xlab,ylab,main,pch,ylim,xlim | various graphical parameters. |
| ... | further graphical parameters for the plot. |

### Details

The cs argument should be a value from the variable set to the cs argument in the amelia function for this output. This function will not work if the ts and cs arguments were not set in the amelia function.

## 7.11 write.amelia: Write Amelia imputations to file

### Description

Writes the imptuted datasets to file from a run of amelia.

### Usage

```
write.amelia(obj, file.stem, extension = NULL, format = "csv", ...)
```

### Arguments

| | |
|---|---|
| obj | an object of class "amelia"; typically output from the function amelia. |
| file.stem | the leading part of the filename to save to output The imputation number and extension will be added to complete the filename. This can include a directory path. |
| extension | the extension of the filename. This is simply what follows file.stem and the imputation number. |
| format | one of the following output formats: csv, dta or table. See details. |
| ... | further arguments for the write functions. |

### Details

write.amelia writes each of the imputed datasets to a file using one of the following functions: write.csv, write.dta, or write.table. You can pass arguments to these functions from write.amelia.

If you were to set file.stem to "outdata" and the extension to ".csv" , then the resulting filename of the written files will be

```
outdata1.csv
outdata2.csv
outdata3.csv
...
```

and so on.

### See Also

write.csv, write.table, write.dta

# References


Dempster, Arthur P., N.M. Laird and D.B. Rubin. 1977. "Maximum Likelihood Estimation from Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Association* 39:1–38.

Honaker, James, Anne Joseph, Gary King, Kenneth Scheve and Naunihal Singh. 1998-2002. "AMELIA: A Program for Missing Data.". http://gking.harvard.edu/amelia.

Honaker, James and Gary King. 2009. "What to do About Missing Values in Time Series Cross-Section Data.". http://gking.harvard.edu/files/abs/pr-abs.shtml.

King, Gary. 1989. *Unifying Political Methodology: The Likelihood Theory of Statistical Inference*. Ann Arbor: Michigan University Press.

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.

Milner, Helen and Keiko Kubota. 2005. "Why the move to free trade? Democracy and trade policy in the developing countries." *International Organization* 59(1):107–143.

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

Schafer, Joseph L. and Maren K. Olsen. 1998. "Multiple Imputation for multivariate Missing-Data Problems: A Data Analyst's Perspective." *Multivariate Behavioral Research* 33(4):545–571.