Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

Multiple Imputation with Diagnostics (mi) in R: Opening Windows into the Black Box

Yu-Sung Su
Columbia University
and
New York University

Andrew Gelman
Columbia University

Jennifer Hill
New York University

Masanao Yajima
University of California
at Los Angeles

Abstract

Our **mi** package in R has several features that allow the user to get inside the imputation process and evaluate the reasonableness of the resulting models and imputations. These features include: flexible choice of predictors, models, and transformations for chained imputation models; binned residual plots for checking the fit of the conditional distributions used for imputation; and plots for comparing the distributions of observed and imputed data in one and two dimensions. In addition, we use Bayesian models and weakly informative prior distributions to construct more stable estimates of imputation models. Our goal is to have a demonstration package that (a) avoids many of the practical problems that arise with existing multivariate imputation programs, and (b) demonstrates state-of-the-art diagnostics that can be applied more generally and can be incorporated into the software of the others.

Keywords: multiple imputation, model diagnostics, chained equations, weakly informative prior, **mi**, R.

1. Introduction

The general statistical theory and framework for managing missing information has been well developed since Rubin (1987) published his pioneering treatment of multiple imputation methods for nonresponse in surveys. Several software packages have been developed to implement these methods to deal with incomplete datasets. However, each of these imputation packages is, to a certain degree, a black box, and the user must trust the imputation procedure without much control over what goes into it and without much understanding of what comes out.

Model checking and other diagnostics are generally an important part of any statistical procedure. Examining the implications of imputations is particularly important because of the inherent tension of multiple imputation: that the model used for the imputations is not in general the same as the model used for the analysis (Meng 1994; Fay 1996; Robins and Wang 2000). We have created an open-ended, open source **mi** package, not only to solve these imputation problems, but also to develop and implement new ideas in modeling and model checking.

Our **mi** package in R (R Development Core Team 2009) has several features that allow the user to get inside the imputation process and evaluate the reasonableness of the resulting model and imputations. These features include: flexible choice of predictors, models, and transformations for chained imputation models; binned residual plots for checking the fit of the conditional distributions used for imputation; and plots for comparing the distributions of observed and imputed data in one and two dimensions. **mi** uses an algorithm known as a chained equation approach (van Buuren and Oudshoorn 2000; Raghunathan, Lepkowski, Van Hoewyk, and Solenberger 2001); the user specifies the conditional distribution of each variable with missing values conditioned on other variables in the data, and the imputation algorithm sequentially iterates through the variables to impute the missing values using the specified model.

This article will spare readers from reading through the theoretical background of multiple imputation (Little and Rubin 2002; Gelman and Hill 2007, chapter 25). Rather, the major goal is to demonstrate the way in which the users can perform multiple imputation with **mi** and to introduce functions for diagnostics after imputation. The paper proceeds as follows: In Section 2, we provide a simplified overview of steps to do sensible multiple imputation. In Section 3, we demonstrate some novel features and functions of **mi** that solve some imputation problems that have not been addressed by other software. These features include: (1) Bayesian regression models to address problems with separation; (2) imputation steps that deal with semi-continuous data; (3) modeling strategies that handle issues of perfect correlation and structural correlation; (4) functions that check the convergence of the imputations; and (5) plotting functions that visually check the imputation models. In Section 4, we demonstrate how to apply these functions using an example of a study of people living with HIV in New York City (Messeri, Lee, Abramson, Aidala, Chiasson, and Jessop 2003). In Section 5, we discuss some loose ends and future plans for our **mi** package.

2. Basic Setup of mi

The procedure to obtain sensible multiply imputed datasets approach requires roughly four steps: setup, imputation, analysis, and validation. Each step is divided into substeps as follows:

1. Setup
   - Display of missing data patterns.
   - Identifying structural problems in the data and preprocessing.
   - Specifying the conditional models.
2. Imputation
   - Iterative imputation based on the conditional model.

- Checking the fit of conditional models.
- Checking the convergence of the procedure.
- Checking to see if the imputed values are reasonable.

3. Analysis
    - Obtaining completed data.
    - Pooling the complete case analysis on multiply imputed datasets.
4. Validation
    - Sensitivity analysis.
    - Cross validation.

At first glance, it may seem more complicated to conduct multiple imputation using **mi** because we outline steps that other packages ignore. However, in Section 4, we will demonstrate the way in which users can easily implement these imputation steps using **mi** via an example. **mi** is designed for both novice and experienced users. For the novice users, **mi** has a step-by-step interactive interface where users choose options from the given multiple choices (see Section 4.6). For more experienced users, **mi** has simple commands that users can use to conduct a multiple imputation.

The implementation of the **mi** package is straightforward. The core function is a generic function `mi(object, ...)` which implements one of two methods depending on whether the input is of the the `data.frame` class or the S4 class `mi`. The `mi` class defines the output returned by `mi()` when it finishes a multiple imputation with a dataset. The usages of the two methods are described below:

```
## S4 method for signature 'data.frame':
 mi(object, info, n.imp = 3, n.iter = 30,
    R.hat = 1.1, max.minutes = 20, rand.imp.method = "bootstrap",
    preprocess = TRUE, run.past.convergence = FALSE,
    seed = NA, check.coef.convergence = FALSE,
    add.noise = noise.control(), post.run = TRUE)

## S4 method for signature 'mi':
 mi(object, info, n.iter = 30, R.hat = 1.1,
    max.minutes = 20, rand.imp.method = "bootstrap",
    run.past.convergence = FALSE, seed = NA)
```

- `object`: A data frame or an `mi` object that contains an incomplete data. **mi** identifies `NA`'s as the missing data.
- `info`: The `mi.info` object (see Section 2.1).
- `n.imp`: The number of multiple imputations. Default is 3 chains.
- `n.iter`: The maximum number of imputation iterations. Default is 30 iterations.

- `R.hat`: The value of the $\hat{R}$ statistic used as a convergence criterion. Default is 1.1 (Gelman, Carlin, Stern, and Rubin 2004).
- `max.minutes`: The maximum minutes to operate the whole imputation process. Default is 20 minutes.
- `rand.imp.method`: The methods for random imputation. Currently, `mi()` implements only the `boostrap` method.
- `preprocess`: Default is `TRUE`. `mi()` will transform the variables that are of `nonnegative`, `positive-continuous`, and `proportion` types (see Section 3.2).
- `run.past.convergence`: Default is `FALSE`. If the value is set to be `TRUE`, `mi()` will run until the values of either `n.iter` or `max.minutes` are reached even if the imputation is converged.
- `seed`: The random number seed.
- `check.coef.convergence`: Default is `FALSE`. If the value is set to be `TRUE`, `mi()` will check the convergence of the coefficients of imputation models.
- `add.noise`: A list of parameters for controlling the process of adding noise to `mi()` via `noise.control()` (see Section 3.3.2).
- `post.run`: Default is `TRUE`. `mi()` will run 20 more iterations after an imputation process is finished if and only if `add.noise` is not `FALSE`. This is to mitigate the influence of the noise to the whole imputation process.

`mi()` is a wrapper of several key components:

2.1. Imputation Information Matrix

`mi.info()` produces a matrix of imputation information that is needed to impute the missing data. After the information is extracted from a dataset, users can still alter default model specifications that are automatically created using this imputation information. Such a matrix of imputation information allows the users to have control over the imputation process. It contains the following information:

- `name`: The names of variables in the dataset.
- `imp.order`: A vector that records the order of each variable in the iterative imputation process. If such a variable is missing for all the observations (see `all.missing`), the `imp.order` slot will record an `NA` (see Section 3.3).
- `nmis`: A vector that records the number of data points that are missing in each variable.
- `type`: A vector that contains the information of the variable types which are determined by `typecast()` (see Section 2.2).
- `var.class`: A vector that records the classes of the input variables.
- `level`: A list of the levels of the input variables.

- include: A vector of indicators that decide whether or not (Yes/No) to include a specific variable in an imputation process. If include is No, the variable will not show up either as a predictor or as a variable to be imputed.
- is.ID: A vector of indicators that determine whether or not (Yes/No) a specific variable is an identification (ID) variable. If a variable is detected as an ID variable, it will not be included in the imputation process; thus the include slot records a No value. ID variables are usually not problematic as dependent variables, since in most of the cases, they have no missing values. But when they are included in a model as predictors, they induce an unwanted order effect of the data into the model (unless the data is a repeated measure study and ID variables are treated as categorical variables). However, because ID variables are hard to detect, users should carefully check to see if all such variables have been detected.
- all.missing: A vector of indictors that identify whether or not (Yes/No) a variable is missing for all the observation. If the value is TRUE, such a variable will be excluded in the imputation process because it is not possible to impute sensible values. The include slot records a No value if all.missing is TRUE.
- collinear: A vector of indicators that shows whether or not (Yes/No) a variable is perfectly collinear with another variable. If the value is TRUE, such a variable will be excluded in the imputation process (thus the include slot records a No value) if and only if these two variables have the same missing data pattern.
- imp.formula: A vector of formulas that records the imputation formulas used in the imputation models.
- determ.pred: The deterministic values from a corresponding correlated variable (see Section 3.3).
- params: A list of parameters to pass on to the imputation models.
- other: Other options. This is currently not used.

Users can alter the output of the mi.info matrix using update(). Or it can be done interactively with mi.interactive(), which is an interactive version of mi() (see Section 4.6). For instance, if we have a variable x in a dataset, and we do not want to include it in the imputation process, we can update the include slot of the mi.info matrix by:

```
R> info

  names include order number.mis all.mis       type correlated
1     x     Yes     1          3      No continuous         No
...

R> info <- update(info, "include", list("x" = FALSE))
R> info

  names include order number.mis all.mis       type correlated
1     x      No     1          3      No continuous         No
```

2.2. Variable Types

mi handles eleven variable types. `mi()` uses `typecast()` to automatically identify eight different variable types. `mi.preprocess()` specifies the `log-continuous` type via transformation (see Section 3.2). The variable types that are not automatically identified by `typecast()` are `count` and `predicative-mean-matching` type.[1] These two types must be user-specified via `update()`. `typecast()` identifies variable types using the rules depicted in Figure 1.

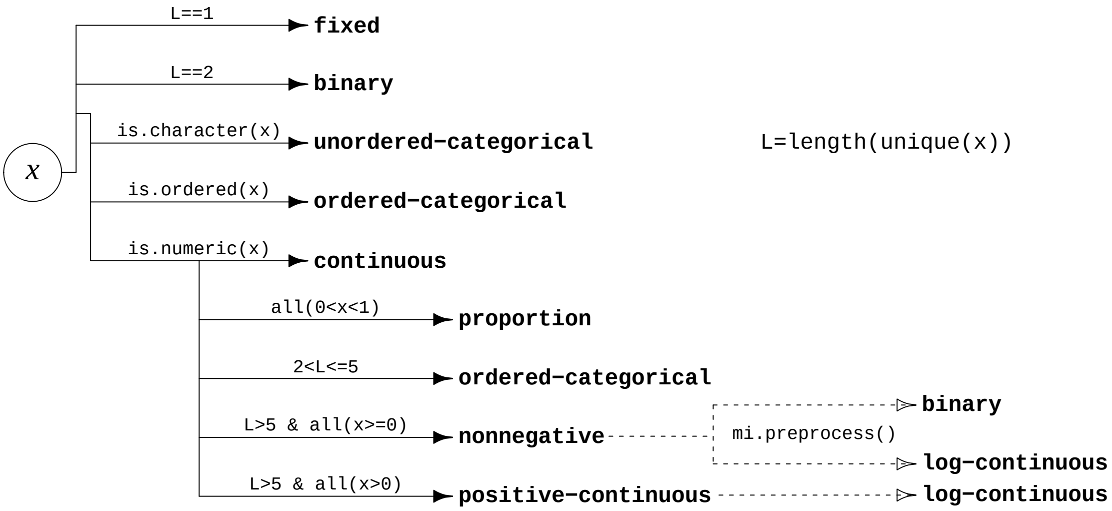


Figure 1: Illustration of the rules of `typecast()` to identify and classify different variable types. **mi** currently handles eleven variable types: `fixed`, `binary`, `unordered-categorical`, `ordered-categorical`, `proportion`, `positive-continuous`, `nonnegative`, `continuous`, `log-continuous`, `count`, and `predicative-mean-matching`. But **mi** can automatically identify the first eight variable types. `mi.preprocess()` specifies `log-continuous` variable type via transformation. Users have to specify `count` and `predicative-mean-matching` variable types manually via `update()` or interactively via `mi.iteractive()`.

The rules, which `typecast()` uses to identify each variable type, are listed as follows:

1. `fixed`: Any variable that contains a single unique value.
2. `binary`: Any variable that contains two unique values.
3. `ordered-categorical`: Any variable that has the ordered attribute that is determined by `is.ordered()` in R. Or any numerical variable that has 3 to 5 unique values.
4. `unordered-categorical`: Any factor or character variable that is determined by `is.factor()` and `is.character()` in R.
5. `proportion`: Any numerical variable that has its values fall between 0 and 1, not including 0 and 1.

[1] `predictive-mean-matching` is not really a variable type. We include it here to invoke `mi()` to fit a model using the predictive mean matching method (see Section 2.3)

6. `positive-continuous`: Any numerical variable that is always positive and has more than 5 unique values.

7. `nonnegative`: Any numerical variable that is always nonnegative and has more than 5 unique values.

8. `continuous`: Any numerical variable that is modeled as continuous without transformation.

9. `log-continuous`: log-scaled continuous variable, specified by `mi.preprocess()` (see Section 3.2).

10. `count`: An user-specified variable type.

11. `predictive-mean-matching`: User-specified (see Section 2.3).

Once the variable type is determined by `typecast()`, the `type` information will be stored in the `mi.info` matrix. Nonetheless, users can alter this default judgment. For instance, if `riot` is the number of riots in a specific year, its values are very likely to fall between 0 and any positive integer. Hence, `typecast()` is going to identify `riot` as either `ordered-categorical`, `positive-continuous` or `nonnegative` type, depending on number of unique values it has and whether or not its values contains 0. You can alter this judgment by updating the `type` slot in a `mi.info` matrix as:

```
R> info

  names include order number.mis all.mis        type correlated
1  riot     Yes    1          23      No nonnegative         No
...

R> info <- update(info, "type", list("riot" = "count"))
R> info

  names include order number.mis all.mis  type correlated
1  riot     Yes    1          23      No count         No
...
```

2.3. Imputation Models

By default, **mi** chooses the conditional models via `type.model()`, a function that determines which imputation models to use based on the variable types determined by `typecast()`. Table 1 lists the default regression models corresponding to variable types:

mi uses `mi.continuous()` to impute `positive-continuous`, `nonnegative`, and `proportion` variable types. This could lead to improper imputations because these variable types are of non-standard distributions. For `positive-continuous` and `nonnegative` variable types, **mi** offers an option to impute these data using the predictive mean matching method (Rubin 1987). When imputing for a variable x_1 using variables $x_2, \dots, x_k$ as predictors, the predictive mean matching method starts by calculating the posterior mean of the x_1 given the other

Table 1: List of regression models, corresponding to variable types

Variable Types	Regression Models
`binary`	`mi.binary`
`continuous`	`mi.continuous`
`count`	`mi.count`
`fixed`	`mi.fixed`
`log-continuous`	`mi.continuous`
`nonnegative`	`mi.continuous`
`ordered-categorical`	`mi.polr`
`unordered-categorical`	`mi.categorical`
`positive-continuous`	`mi.continuous`
`proportion`	`mi.continuous`
`predictive-mean-matching`	`mi.pmm`

predictors and the parameters (the mean of the posterior predictive distribution). Then imputations are created by finding, for each observation with the closest predictive mean. The observed value from this "match" is used as the imputed value. This method can be helpful when imputing data from distributions that are more difficult to model directly. Nonetheless, this method can fail when rates of missingness are high or when the missing values fall outside the range of the observed data. Hence, we have designed a new way to model these types of data via transformation (see Section 3.2).

3. Novel Features

Our **mi** has some novel features that solve some open issues in multiple imputation.

3.1. Bayesian Models to Address Problems with Separation

Logistic regression, and discrete data models more generally, commonly suffer from the problem of separation. This problem occurs whenever the outcome variable is perfectly predicted by a predictor or a linear combination of the predictor variables. This can happen even with a modest number of predictors, particularly if the proportion of "successes" in the response variable is relatively close to 0 or 1. The risk of separation typically increases as the number of predictors increases. However, multiple imputation is generally strengthened by including many variables, which can help to impute more precisely and also may help to satisfy the missing at random assumption. When imputing large-scale surveys to create public-use multiply imputed datasets, several hundred variables might need to be imputed. And it is unclear how or if we should start discarding subsets of variables from some or all of the conditional models. Separation problems can cause the chained equation algorithms to either fail or impute unreasonable values.

To address problems with separation, we have augmented our **mi** to allow for Bayesian versions of generalized linear models (as implemented in the functions `bayesglm()` and `bayespolr()`) that automatically handle separation (Gelman, Jakulin, Pittau, and Su 2008). These are set up for different imputation models (see Table 2).[2]

[2] We are working on Bayesian versions of multinomial models for unordered categorical variables. `mi()` now

Table 2: Lists of Bayesian Generalized Linear Models Used in **mi** Regression Functions

mi Functions	Bayesian Functons
`mi.continuous()`	`bayesglm()` with `gaussian` family
`mi.binary()`	`bayesglm()` with `binomial` family (default uses `logit` link)
`mi.count()`	`bayesglm()` with `quasi-poisson` family (overdispersed poisson)
`mi.polr()`	`bayespolr()`

3.2. Imputing Semi-Continuous Data with Transformation

Semi-continuous data (`positive-continuous`, `nonnegative` and `proportion` variable types in **mi**) are completely straightforward to impute and are typically not modeled in a reasonable way in other imputation software. The difficulty comes from the fact that these kinds of data have bounds or truncations and are not of standard distributions. Our algorithm models these data with transformation via `mi.preprocess()`. By default, this transformation is automatically done in `mi()` by setting the option `preprocess = TRUE`. Users can also transform the data using `mi.preprocess()` and feed it back into `mi()`. For instance,

```
R> newdata <- mi.preprocess(data)
R> IMP <- mi(newdata, preprocess = FALSE)
```

For the `nonnegative` variable type, `mi.preprocess()` creates two ancillary variables. One is an indicator for which values of the `nonnegative` variable are bigger than 0. The other ancillary variable takes the log of such a variable on any value that is bigger than 0. For the `positive-continuous` variable type, `mi.preprocess()` takes the log of such a variable. For the `proportion` variable type, `mi.preprocess()` does a logit transformation on such a variable as $\text{logit}\,(x) = \log\left(\frac{x}{1-x}\right)$.

Figure 2 illustrates this transformation process. Users can transform the data back to its original scale using `mi.postprocess()`. This is implemented automatically in `mi.completed()` (see Section 4.4.2). For the `positive-continuous` variable (`x1`), it is going to be transformed back as $\widehat{\texttt{x1}} = \exp(\texttt{x1}) \times \texttt{x1.ind}$; for the `nonnegative` variable (`x2`), $\widehat{\texttt{x2}} = \exp(\texttt{x2})$; and for the `proportion` variable (`x3`), $\widehat{\texttt{x3}} = \text{logit}^{-1}\,(\texttt{x3})$.

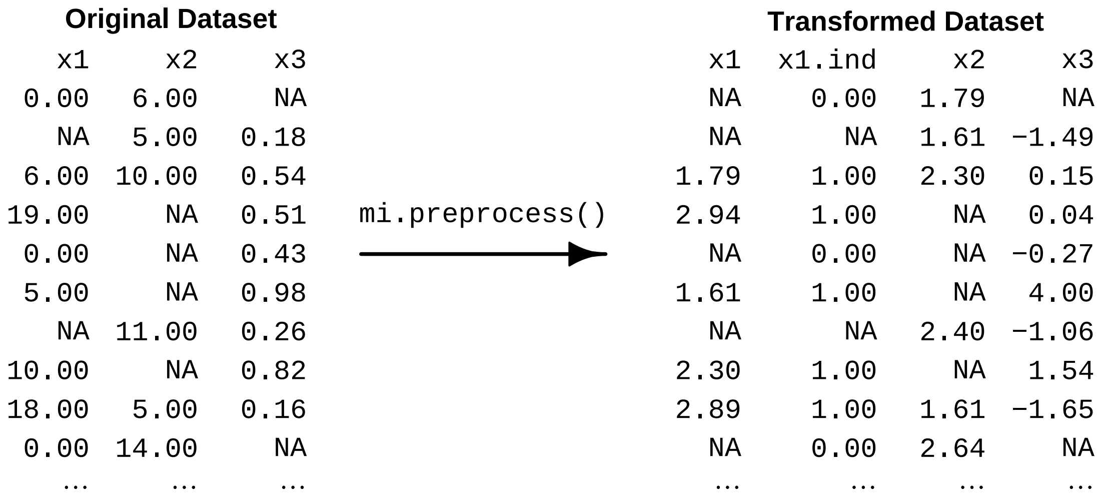
3.3. Imputing Data with Collinearity

mi can deal with two types of data with collinearity. One is for data with perfect correlation of two variables (e.g., $x_1 = 10x_2 - 5$). The other one is for data with additive constraints across several variables (e.g., $x_1 = x_2 + x_3 + x_4 + x_5 + 10$).

Perfect Correlation

In real datasets, a variable may appear in a dataset multiple times or with different scale. For example, GDP per capita and GDP per capita in thousand dollars could both be in a dataset. For these variables, if the missingness pattern of these two variables are the same, `mi()` will include only one of the duplicated variables in the iterative imputation process. To

uses `mi.categorical()`, which uses `multinom()` (multinomial log-linear model) (Venables and Ripley 2002), to handle with unordered categorical variables.



Original Dataset

```
   x1     x2    x3
 0.00   6.00    NA
   NA   5.00  0.18
 6.00  10.00  0.54
19.00     NA  0.51
 0.00     NA  0.43
 5.00     NA  0.98
   NA  11.00  0.26
10.00     NA  0.82
18.00   5.00  0.16
 0.00  14.00    NA
  ...    ...   ...
```

mi.preprocess()

Transformed Dataset

```
  x1  x1.ind    x2     x3
  NA    0.00  1.79     NA
  NA      NA  1.61  -1.49
1.79    1.00  2.30   0.15
2.94    1.00    NA   0.04
  NA    0.00    NA  -0.27
1.61    1.00    NA   4.00
  NA      NA  2.40  -1.06
2.30    1.00    NA   1.54
2.89    1.00  1.61  -1.65
  NA    0.00  2.64     NA
 ...     ...   ...    ...
```

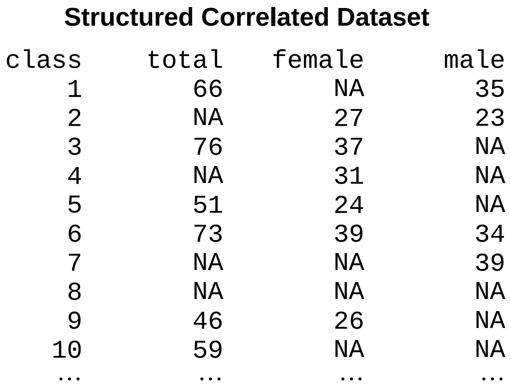
Figure 2: Illustration of the way in which `mi.preprocess()` transforms the data of `nonnegative` (x1), `positive-continuous` (x2) and `proportion` (x3) variable types.

impute data with such a perfect correlation, `mi()` will firstly identify such a pair of correlated variables via `mi.info()` and exclude one of them from the imputation process. Then `mi()` will use `mi.copy()` to impute the missing values of the excluded variable by duplicating values from the correlated variable. If for some reasons, the duplicated variables do not have the same missingness pattern, `mi()` keeps them in the iterative imputation process and utilizes the technique stated in Section 3.3.2 to impute them.

Additive Constraints

General additive constraints can be hard to address with the chained equation algorithm because they can be more difficult to identify. This problem can exist either in deterministic situations (such as inclusion of all the items in a scale as well as the total scale score) or situations included in the model that by chance a subset of variables end up being functionally dependent on each other (i.e., each is a linear combination of the rest). Figure 3 displays a hypothetical example of a dataset with additive constraints. In this example, we have the number of female students (`female`), the number of male students (`male`) and the total number of students (`total`) of different classes. Hence, `total` = `female`+`male`. Such a problem could easily be dealt with if an investigator spots such a problem beforehand and takes out one of the variables. However, such situations are often overlooked in practice. Moreover, if such a problem of additive constraints exists across many variables, particularly if they are not logically related and have no explicit variable names attached to it, this problem could easily remain undetected.

The problem of additive constraint is exacerbated when the missingness rate is high. And the identification problem arises when more than two variables are missing. Take class 4 in the dataset for an example. `mi()` will first randomly assign a value for `male` in the class 4, say 20. Then `mi()` imputes `total` by regressing it on `male` and `female`. In this case, the value would be 51. Using the imputed value for `total`, the next imputation of `mi()` for `male` will be 20. And using this imputed value for `male`, the next imputed value for `total` will be 51. This situation will continue to repeat like this over and over. The result of this problem is that `mi()` will not be able to explore the entire response surface of the imputed variables.



Structured Correlated Dataset

class	total	female	male
1	66	NA	35
2	NA	27	23
3	76	37	NA
4	NA	31	NA
5	51	24	NA
6	73	39	34
7	NA	NA	39
8	NA	NA	NA
9	46	26	NA
10	59	NA	NA
...	...	...	...

Figure 3: Illustration of a dataset with additive constraints. In this dataset, `total` = `female`+ `male`.

To deal with this problem, we introduce an artificial set of prior distributions into the iterative imputation process. The purpose is to create noise that breaks the additive structure so as to make `mi()` explore more of the response surface of the imputed variables. At the same time, because the priors originate from the observed data, they also ensure that the imputed values do not deviate too far from the observation. `mi()` currently offers two options via `noise.control()`, each of which temporarily adds prior information to the model fits.

- Reshuffling noise: By default, `mi()` adds noise to the iterative imputation process by randomly imputing missing data from the marginal distribution. In other words, `mi()` imputes the missing data with values not from conditional models but from the observed data randomly with probability p, where p is determined by a constant K, specified by users. In every iteration, `mi()` decides whether or not to impute the missing data from the marginal distribution based on q, where `q = rbinom(n = 1, size = 1, prob = p)`. If `q = 1`, `mi()` imputes the missing data with values from the observed data. Otherwise, it imputes missing data with values from the conditional models.

  The influence of the noise gradually declines because p is gradually decreasing to the zero as the number of iterations increases according to the function of $\frac{K}{s}$. s is the number of imputation iteration. This means `mi()` eventually imputes the missing data only from the conditional models. Hence, depending on the size of K, users can control how much power they want the noise to insert into the iterative imputation process. The standard usage is as below:

  ```
  R> mi(data, add.noise = noise.control(method = "reshuffling", K = 1))
  ```

- Fading empirical noise. In each iteration, `mi()` augments the data by `pct.aug = 10` percent (default) of the completed data by randomly adding new data that are drawn from the observed data. Thus if a completed dataset has 250 data points, `mi()` will augment such a dataset with 25 new data points from the observed data of the complete case. The standard usage is as below:

```
R> mi(data, add.noise = noise.control(method = "fading", pct.aug = 10))
```

By default, `mi()` uses the reshuffling noise. If users have the faith on their data having neither of the two correlation problems, they can choose not to add noise into the imputation process by specifying `mi(data, add.noise = FALSE, ...)`. If any of the two methods of adding noise is used, by default, `mi()` will run 20 more iterations (controlled by `post.run`, default is `TRUE`) without adding any noise to mitigate the influence of the noise.

3.4. Checking the Convergence of the Imputations

Our **mi** offers two ways to check the convergence of the multiple imputation procedure. By default, `mi()` monitors the mixing of each variable by the variance of its mean and standard deviation within and between different chains of the imputation. If the $\hat{R}$ statistic is smaller than 1.1, (i.e., the difference of the within and between variance is trivial), the imputation is considered converged (Gelman, Carlin, Stern, and Rubin 2004). Additionally, by specifying `mi(data, check.coef.convergence = TRUE, ...)`, users can check the convergence of the parameters of the conditional models.

3.5. Model Checking and Other Diagnostic for the Imputations Using Graphics

Model checking and other diagnostics are generally an important part of any statistical procedure. This is particular important to imputation because the model used for imputation in general is not the same as the model used for the analysis. Yet, there is noticeable dearth of such checks in the multiple imputation world. Thus imputations are, to a certain degree, a black box. The lack of development in imputation diagnostics comes from that fact that the ways in which to evaluate the adequacy of imputed values that are proxies for data points are by definition unknown. Our **mi** addresses this problem with three strategies.

- Imputations are typically generated using models, such as regressions or multivariate distributions, which are fit to observed data. Thus the fit of these models can be checked (Gelman, Van Mechelen, Verbeke, Heitjan, and Meulders 2005).

- Imputations can be checked using a standard of reasonability: the differences between observed and missing values, and the distribution of the completed data as a whole, can be checked to see whether they make sense in the context of the problem being studied (Abayomi, Gelman, and Levy 2008).

- We can use cross-validation to perform sensitivity analysis to violations of our assumptions. For instance, if we want to test the assumption of missing at random, after obtaining the completed dataset (original data plus imputed data) using **mi**, we can randomly create missing values on these imputed datasets and re-impute the missing data (Gelman, King, and Liu 1998). By comparing the imputed dataset before and after this test, we can assess the validity of the missing at random assumption.

So far, **mi** only implements the first two solutions with various plotting functions. We demonstrate the usages of these functions in Section 4.3.2.

4. Example

In this Section, we will demonstrate some basic steps of **mi** with an example.

4.1. A Study of HIV-Positive People in New York City

The `CHAIN` dataset included in **mi** is a subset of the Community Health Advisory and Information Network (`CHAIN`) study. This study is a longitudinal cohort study of people living with HIV in New York City and is conducted by Columbia University School of Public Health (Messeri, Lee, Abramson, Aidala, Chiasson, and Jessop 2003). The `CHAIN` cohort was recruited in 1994 from a large number of medical care and social service agencies serving HIV in New York City. Cohort members were interviewed up to 8 times through 2002. A total of 532 CHAIN participants completed at least one interview at either the 6th, 7th or 8th rounds of interview, and 508, 444, 388 interviews were completed respectively at rounds 6, 7 and 8 (CHAIN 2009). For simplicity, our analysis here discards the time aspect of the dataset and use only the 6th round of the survey. The dataset has 532 observations and has the following 8 variables:

- `h39b.W1`: Log of self reported viral load level, 0 represents undetectable level.
- `age.W1`: The respondent's age at time of interview.
- `c28.W1`: The respondent's family annual income. Values range from under $\$5,000$ to $\$70,000$ or over.
- `pcs.W1`: A continuous scale of physical health with a theoretical range between 0 and 100 (better health is associated with higher scale values).
- `mcs37.W1`: A dichotomous measure of poor mental health: 0=No, 1=Yes.
- `b05.W1`: Ordered interval for the CD4 count (the indicator of how much damage HIV has caused to the immune system).
- `haartadhere.W1`: A three-level-ordered variable: 0 = not currently taking highly active antiretroviral therapy (HAART), 1 = taking HAART nonadherent, 2 = taking HAART adherent.

To use the data, users must first load the **mi** library:[3]

```
R> library(mi)
```

[3] The printout of the loaded information shows that **mi** depends upon on several R packages, including **MASS** and **nnet** (Venables and Ripley 2002), **car** (Fox 2009), **arm** (Gelman, Su, Yajima, Hill, Pittau, Kerman, and Zheng 2009), **Matrix** (Bates and Maechler 2009), **lme4** (Bates, Maechler, and Dai 2008), **R2WinBUGS** (Sturtz, Ligges, and Gelman 2005), **coda** (Plummer, Best, Cowles, and Vines 2009) and **abind** (Plate and Heiberger 2004). In the last line of the loaded information, R prints out the version number of **mi**. Users are welcome to report bugs or make suggestions to us with the attached version number.

```
Loading required package: MASS
Loading required package: nnet
Loading required package: car
Loading required package: arm
Loading required package: Matrix
...
Loading required package: lme4

...
Loading required package: abind

mi (Version 0.07-1, built: 2009-4-28)
```

Then load the CHAIN dataset in the memory:

```
R> data(CHAIN)
```

4.2. Setup

The first thing to do is to set up the imputation. As with most statistical procedure, one must start with some preliminary analysis to avoid trivial problems. When that is completed, two key steps must be done: choosing the conditional models and specifying the models.

Preliminary analysis is crucial in an iterative procedure such as multiple imputation that uses the chain equation algorithm. Users do not want simple mistakes that arise in the early stages to ruin the end result after a long iteration. In a small dataset, this may not be a serious issue, but for a large dataset, this may be costly. There are problems which **mi** automatically detects. Nonetheless, there are problems that is not possible to be detected automatically by **mi**. For those problems that are difficult to detect, our **mi** will raise flags so that user can keep them in mind.

Display of Missing Data Patterns

Users can get the glimpse of the data by looking at the missingness pattern.

```
R> missing.pattern.plot(CHAIN, gray.scale = TRUE)
```

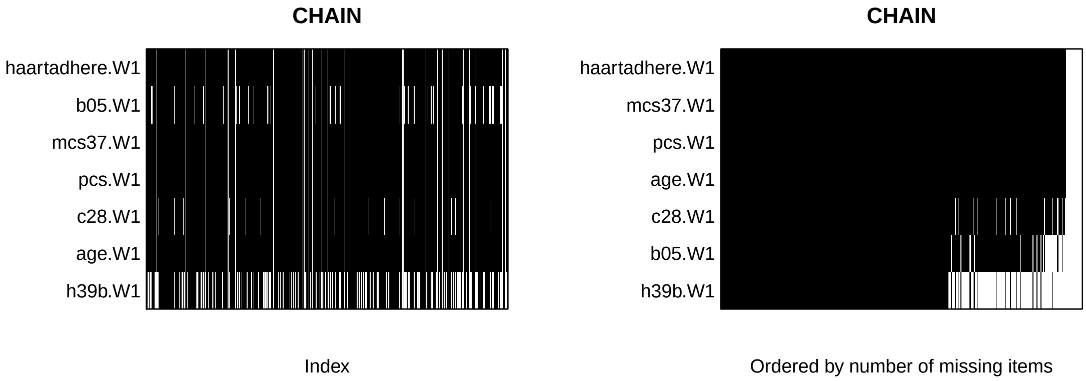
Or simply type:

```
R> mp.plot(CHAIN, gray.scale = TRUE)
```

Figure 4(a) shows the data matrix with observed values in black and missing values in white. At this point it is difficult to detect anything, but we also have the option to order them by the missing data rates (the x-axis is the data index and the y-axis is the variable index).

```
R> mp.plot(CHAIN, y.order = TRUE, x.order = TRUE, gray.scale = TRUE)
```

Figure 4(b) reveals that h39b.W1 variable has the highest missing rate and b05.W1 has the least. Also for all the other variables that have missingness are for the data that have everything missing for all of the variables. When there exist observations with all variables missing,



(a) (b)

Figure 4: Missingness pattern plot. The observed values are plotted with black and the missing values are in white. In the right panel, variables and cases are ordered by proportion missing.

the user may want to consider removing these observations since they will not add any new information to the imputation procedure.

Identifying Structural Problems in the Data and Preprocessing

Before starting to impute the missing data, we can review some basic data information, which will feed into the imputation process, and determine whether or not we want to adjust this information.

```
R> info <- mi.info(CHAIN)
R> info
```

```
           names include order number.mis all.mis                type collinear
1        h39b.W1     Yes     1        179      No         nonnegative        No
2         age.W1     Yes     2         24      No positive-continuous        No
3         c28.W1     Yes     3         38      No positive-continuous        No
4         pcs.W1     Yes     4         24      No positive-continuous        No
5       mcs37.W1     Yes     5         24      No              binary        No
6         b05.W1     Yes     6         63      No ordered-categorical        No
7 haartadhere.W1     Yes     7         24      No ordered-categorical        No
```

By default, `mi.info()` prints out seven out of the fourteen categories of the `mi.info` matrix (see Section 2.1). We can see from this output that `h39b.W1`, `age.W1`, `c28.W1`, and `pcs.W1` are variable types that need special treatment (see Section 3.2).

So to address the variable types that `mi.info()` identifies as requiring special treatment, `mi()` preprocesses the data via `mi.preprocess()` and may change the default judgement returned by `typecase()` (see Section 2.2 and Figure 2). `mi()` returns the new information matrix as the slot `mi.info.preprocessed` when the imputation procedure is finished.

```
R> IMP <- mi(CHAIN, preprocess = TRUE)
R> attr(IMP, "mi.info.preprocessed")
```

```
           names include order number.mis all.mis                type collinear
1        h39b.W1     Yes     1        367      No      log-continuous        No
2         age.W1     Yes     2         24      No      log-continuous        No
3         c28.W1     Yes     3         38      No      log-continuous        No
4         pcs.W1     Yes     4         24      No      log-continuous        No
5       mcs37.W1     Yes     5         24      No              binary        No
6         b05.W1     Yes     6         63      No ordered-categorical        No
7 haartadhere.W1     Yes     7         24      No ordered-categorical        No
8    h39b.W1.ind     Yes     8        179      No              binary        No
```

The new information matrix shows that `h39b.W1`, `age.W1`, `c28.W1`, and `pcs.W1` have been transformed into new variables with different scales and types.

Specifying the Conditional Models

`mi()` chooses the conditional models based on the variable types that are determined by `typecast()` (see Section 2.2). By changing the variable types, `mi()` will choose different conditional models to fit the altered variables. For example, you can change the type of `h39b.W1` from `nonnegative` to `continuous` as:

```
R> info <- mi.info(CHAIN)
R> info

           names include order number.mis all.mis                type collinear
1        h39b.W1     Yes     1        179      No         nonnegative        No
...

R> info.upd <- update(info, "type", list("h39b.W1" = "continuous"))
R> info.upd

           names include order number.mis all.mis                type correlated
1        h39b.W1     Yes     1        179      No          continuous         No
...
```

By default, `mi()` assumes linearity between the outcomes and predictors.

```
R> info$imp.formula

                                                                   h39b.W1
"h39b.W1 ~ age.W1 + c28.W1 + pcs.W1 + mcs37.W1 + b05.W1 + haartadhere.W1"
                                                                    age.W1
"age.W1 ~ h39b.W1 + c28.W1 + pcs.W1 + mcs37.W1 + b05.W1 + haartadhere.W1"
                                                                    c28.W1
"c28.W1 ~ h39b.W1 + age.W1 + pcs.W1 + mcs37.W1 + b05.W1 + haartadhere.W1"
                                                                    pcs.W1
"pcs.W1 ~ h39b.W1 + age.W1 + c28.W1 + mcs37.W1 + b05.W1 + haartadhere.W1"
                                                                  mcs37.W1
"mcs37.W1 ~ h39b.W1 + age.W1 + c28.W1 + pcs.W1 + b05.W1 + haartadhere.W1"
                                                                    b05.W1
```

```
"b05.W1 ~ h39b.W1 + age.W1 + c28.W1 + pcs.W1 + mcs37.W1 + haartadhere.W1"
                                                           haartadhere.W1
"haartadhere.W1 ~ h39b.W1 + age.W1 + c28.W1 + pcs.W1 + mcs37.W1 + b05.W1"
```

If you want to change the fitted formulas by adding interactions or add squared terms, you can alter the `imp.formula` slot of the `mi.info` matrix via `update()` or interactively via `mi.interactive()`:

```
R> info.upd <- update(info, "imp.formula", list("h39b.W1" =
      "h39b.W1 ~ age.W1^2 + c28.W1*pcs.W1 + mcs37.W1 +
        b05.W1 + haartadhere.W1"))
R> info.upd$imp.formula["h39b.W1"]

                                                                  h39b.W1
"h39b.W1 ~ age.W1^2 + c28.W1*pcs.W1 + mcs37.W1 + b05.W1 + haartadhere.W1"
```

4.3. Imputation

Once everything has been setup correctly, actual imputation is simple. However, there are still few things users should check: the fit of the conditional models and convergence of the imputation algorithm. Diagnostic tools are integrated as parts of `mi()`, but decisions about how to use the diagnostic information must be made by the users. We will provide general guidelines here.

Iterative Imputation Based on the Conditional Model.

`mi()` imputes the missing values based on the conditional models. As demonstrated in the previous sections, you can modify the `mi.info` object and pass it into `mi()` to alter these model settings . If no `mi.info` object is passed into `mi()`, `mi()` will call `mi.info()` internally and use the default setting. Although this is not recommended, we have made the default as reasonable as possible.

```
R> IMP <- mi(CHAIN)

Beginning Multiple Imputation ( Mon Apr 27 17:04:06 2009 ):
Iteration 1
 Imputation 1 : h39b.W1*  age.W1*  c28.W1*  pcs.W1*  mcs37.W1*  b05.W1*  haartadhere.W1*  h39b.W1.ind*
 Imputation 2 : h39b.W1*  age.W1*  c28.W1*  pcs.W1*  mcs37.W1*  b05.W1*  haartadhere.W1*  h39b.W1.ind*
 Imputation 3 : h39b.W1*  age.W1*  c28.W1*  pcs.W1*  mcs37.W1*  b05.W1*  haartadhere.W1*  h39b.W1.ind*
Iteration 2
 Imputation 1 : h39b.W1   age.W1   c28.W1*  pcs.W1   mcs37.W1   b05.W1   haartadhere.W1*  h39b.W1.ind
 Imputation 2 : h39b.W1*  age.W1*  c28.W1   pcs.W1   mcs37.W1*  b05.W1*  haartadhere.W1   h39b.W1.ind
 Imputation 3 : h39b.W1*  age.W1   c28.W1   pcs.W1*  mcs37.W1   b05.W1*  haartadhere.W1   h39b.W1.ind
...
Iteration 14
 Imputation 1 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 2 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 3 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
mi converged ( Mon Apr 27 17:05:21 2009 )
Beginning Multiple Imputation ( Mon Apr 27 17:05:21 2009 ):
Iteration 15
 Imputation 1 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 2 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 3 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
...
Iteration 34
 Imputation 1 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 2 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 3 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
mi converged ( Mon Apr 27 17:07:09 2009 )
```

By default, `mi()` will preprocess the data (`preprocess = TRUE`), add noise to the fitted models (`add.noise = noise.control(K = 1)`), and run 20 more iterations (`post.run = TRUE`) after the first `mi()` is finished. The star symbols attached to the variable names indicate that noise is added in the imputation models.

There are other options to specify number of iterations (`n.iter`), how long `mi()` should run (`max.minutes`), whether or not `mi` should continue when it converged (`run.past.convergence`), etc (see Section 2 or type `?mi` in the R console for details).

Checking the Fit of Conditional Models and Imputed Values

Imputation may take some time to run, depending on the size of the data. Thus we suggest that users should check the fit of the conditional models by looking at several diagnostic plots before running a longer imputation procedure (Gelman, Van Mechelen, Verbeke, Heitjan, and Meulders 2005; Abayomi, Gelman, and Levy 2008). This can be done by plotting the `mi` object (Figure 5) after a reasonable number of iterations. **mi** provides four different plots to visually inspect the fit of the conditional models.

```
R> plot(IMP)
```

These four plots are histogram that plots histograms of the observed (in darker color), the imputed (in dashed line) and the completed (observed plus imputed, in lighter color) values, residual plot (for categorical variable, residual plot is not displayed) that plots the residuals against the predicted values, binned residual plot that plots the average of residuals in bins against the expected values (Gelman, Goegebeur, Tuerlinckx, and Van Mechelen 2000), and bivariate scatterplot that plots the observed against the predictive values of the observed (in lighter color) and imputed (in darker color) values, overlain with fitted lowess curves (Cleveland 1979).

Figure 5 displays the selected variables using these four diagnostic plots. The histograms show that the imputed values are all within reasonable ranges and do not differ much from the observed values. The residual plots show that there are rooms for improvement on imputation models of `age.W1` and `pcs.W1` as there are a fair amount of residuals that fall outside of the 95% error bounds (the dotted lines). The strong pattern of the residual plot of `c28.W1` arises from the discreteness of such a variable. Therefore, a binned residual plot is a better plot to look at here. The binned residual plots reveal a similar story to that of the residual plots. The points in a residual plot is the average of one bin from the points in a residual point. We are looking for a patternless plot with the zero mean in such a plot. The binned residual plots of `age.W1`, `c28.W1` and `pcs.W1` show no significant problem as almost all points of each variable fall within of the 95% error bounds (the lines with lighter color). The lowess curves of the imputed and the observed values in those bivariate scatterplot demonstrate that the imputed values look similar to the observed ones.

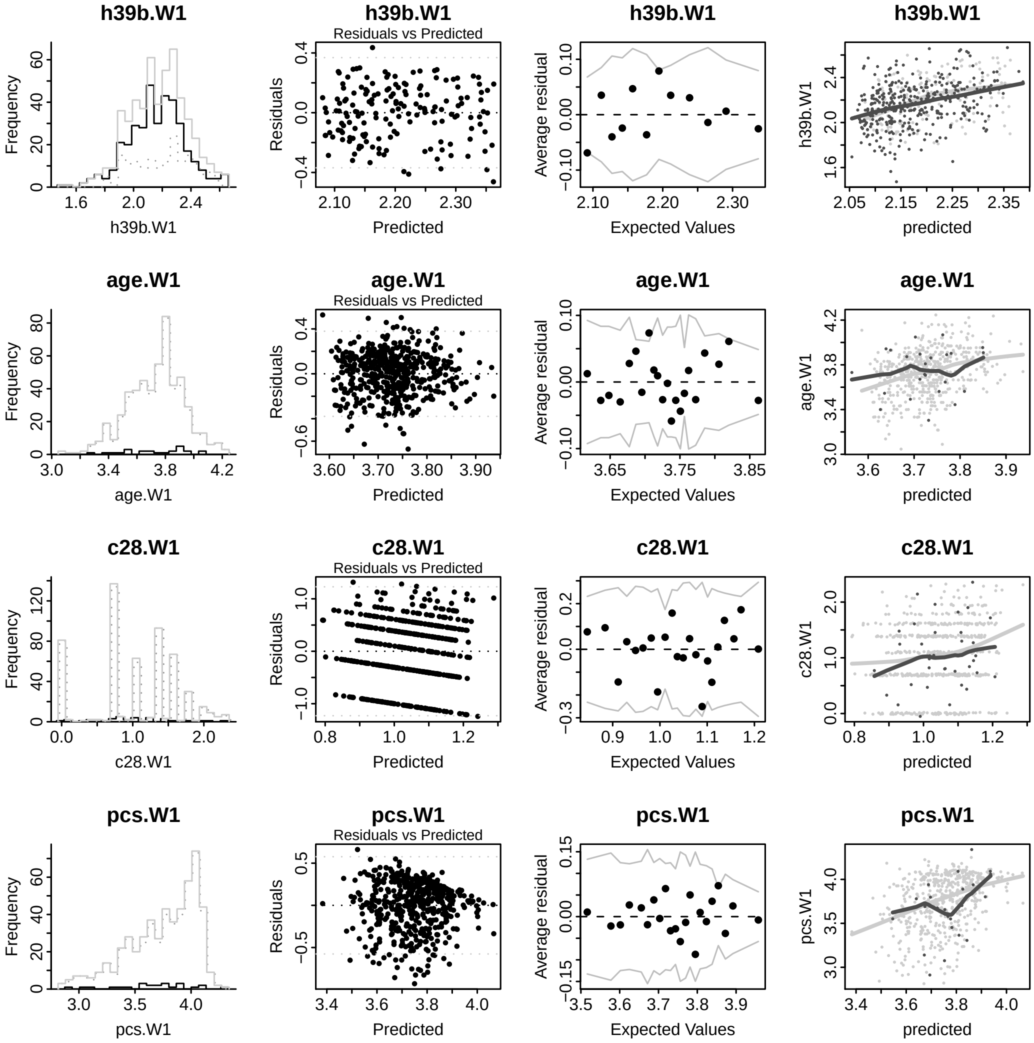


Figure 5: Diagnostic plots for checking the fit of the conditional imputation models. To save page space, only four out of the eight plots for each variable are displayed. The darker color is for the observed value and the lighter color is for the imputed one. By default, these values are plotted against an index number. Plotting against a variable that contains more information is a strongly recommended alternative. Fitted lowess curves are also plotted for both observed and imputed data. A small amount of random noise (jittering) is added to the points so that they do not fall on top of each other.

If users discover a problem when accessing these plots and want to alter the model specification to fix it, they can fix the `mi.info` object via `update()`.

Running Iterative Imputation Longer

When conditional models seem to be fit reasonably, users may want to run the imputation longer. This is achieved by feeding the previous returned `mi` object into `mi()`. Imputation will continue from where it left off. If the previous `mi()` object is converged, you have to specify `run.past.convergence = TRUE` to force `mi()` to run for more iterations.

```
R> IMP <- mi(IMP, run.past.convergence = TRUE, n.iter = 10)
```

```
Beginning Multiple Imputation ( Mon Apr 27 18:05:33 2009 ):
Iteration 35
 Imputation 1 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 2 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 3 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
...
Iteration 44
 Imputation 1 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 2 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
 Imputation 3 : h39b.W1   age.W1   c28.W1   pcs.W1   mcs37.W1   b05.W1   haartadhere.W1   h39b.W1.ind
mi converged ( Mon Apr 27 18:06:31 2009 )
```

Checking the Convergence of the Procedure

Checking the convergence of multiple imputation is still an open research question. By default, `mi()` checks the mean and standard deviation of each variable for different chains. It considers the imputation to have converged when the $\hat{R} < 1.1$ for all the parameters (Gelman, Carlin, Stern, and Rubin 2004). There is a `R.hat` option in `mi()` that allows users to adopt more stringent rule on checking convergence using the $\hat{R}$ statistics (`mi(CHAIN, R.hat = 1)`. Users can also check the convergence of parameters of each conditional model by specifying `mi(CHAIN, check.coef.convergence = TRUE)`. Figure 6 shows that the $\hat{R}$ value of each variable is smaller than 1.1 indicate that the imputation is converged.

```
R> plot(IMP@bugs)
```

4.4. Analysis

One of the nice features of multiply imputed data is that we can conduct analyses as if the data were complete. Results from an analysis performed on each dataset must then be combined in a sensible way, for instance by using formulas proposed by Rubin (1987).

Pooling the Complete Case Analysis on Multiply Imputed Datasets

`mi()` facilitates the analysis process by providing functions that perform these separate analyses and then combine the separate estimates into one estimate and standard error. **mi** currently offers seven regression functions: `lm.mi()`, `glm.mi()`, `polr.mi()`, `bayesglm.mi()`, `bayespolr.mi()`, `lmer.mi()`, and `glmer.mi()`.

```
R> fit <- lm.mi(h39b.W1 ~ age.W1 + c28.W1 + pcs.W1 + mcs37.W1 +
+   b05.W1 + haartadhere.W1, IMP)
R> display(fit)
```

```
=======================================
Pooled Estimate
=======================================
```

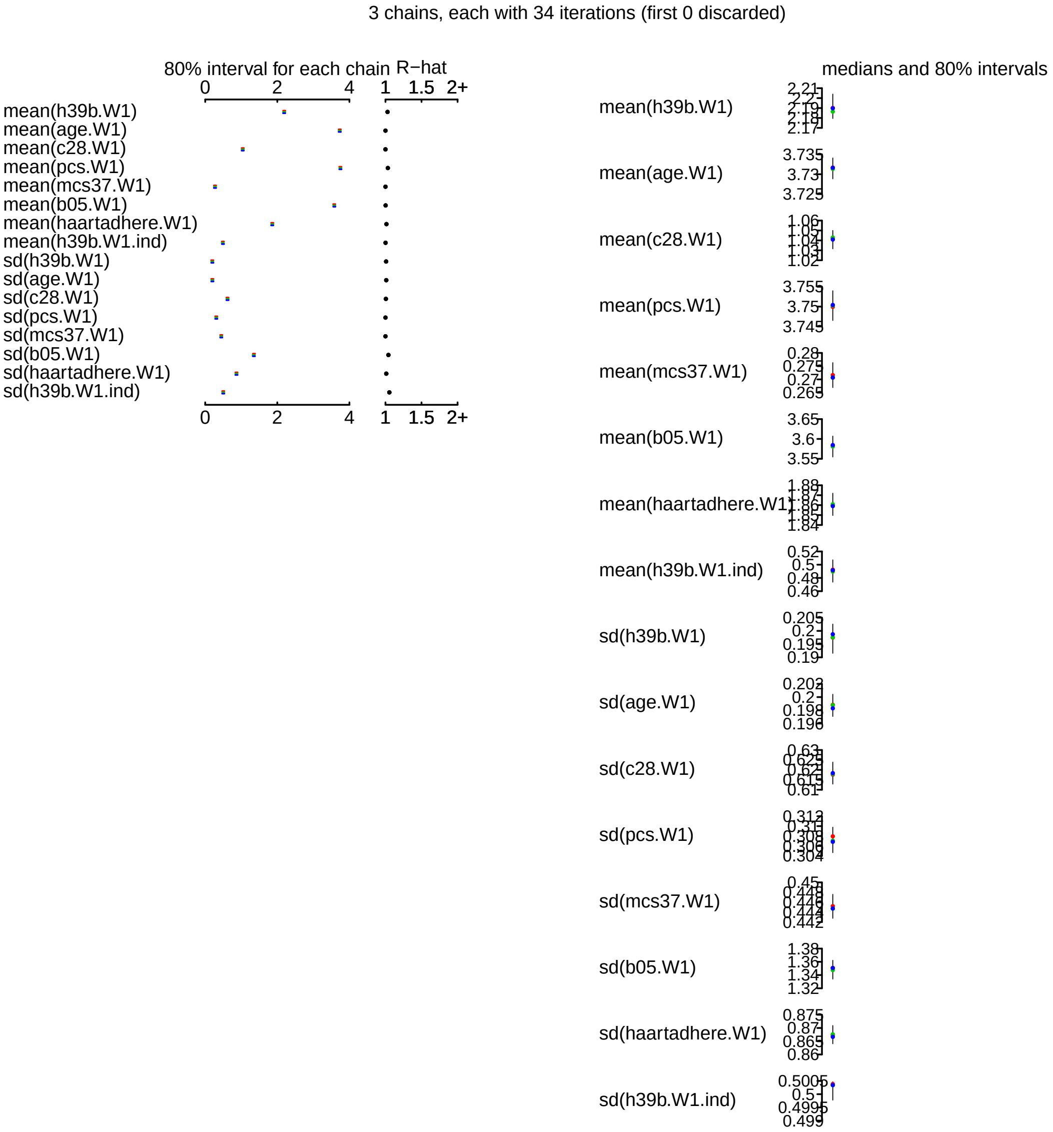


Figure 6: Plot of the summary of the mean and the standard deviation of each variable for the different chains of imputations. All the $\hat{R}$ statistics are smaller than 1.1, indicating the imputation is converged.

```
lm.mi(formula = h39b.W1 ~ age.W1 + c28.W1 + pcs.W1 + mcs37.W1 +
    b05.W1 + haartadhere.W1, mi.object = IMP)
               coef.est coef.se
(Intercept)    16.60     1.45
age.W1         -0.11     0.02
c28.W1         -0.39     0.10
pcs.W1         -0.03     0.02
mcs37.W1        0.80     0.42
```

```
b05.W1         -1.03     0.14
haartadhere.W1 -1.21     0.22

=======================================
Separate Estimate for each Imputation
=======================================

** Imputation 1 **
lm(formula = formula, data = mi.data[[i]])
               coef.est coef.se
(Intercept)    16.27     1.45
age.W1         -0.11     0.02
c28.W1         -0.40     0.10
pcs.W1         -0.03     0.02
mcs37.W1        0.77     0.43
b05.W1         -0.89     0.14
haartadhere.W1 -1.26     0.22
---
n = 532, k = 7
residual sd = 4.31, R-Squared = 0.22

...
```

Obtaining Completed Datasets

If the users prefer to perform separate data analyses for each dataset by themselves, they can easily extract the completed datasets from the `mi` object via `mi.completed()`. This will return a list that contains multiple datasets.

```
R> IMP.dat.all <- mi.completed(IMP)
```

They can extract just one dataset from a specific chain of imputations via `mi.data.frame()`.

```
R> IMP.dat <- mi.data.frame(IMP, m = 1)
```

mi also offers an option to output these multiply imputed datasets into files. Currently, **mi** only supports three data formats: `csv`, `dta`, and `table`. The default output data format is `csv`.

```
R> write.mi(IMP)
```

The output files shall be stored under the working directory. The file names will be `midata1.csv`, `midata2.csv`, `midata3.csv`, …, and so on.

4.5. Validation

The validation step is still under construction. However, we present some ideas of the ways in which users can validate their results obtained from **mi**.

Sensitivity Analysis

Multiple imputation is based on many assumptions, thus it is natural to test how sensitive imputed values are to these assumptions. One of the key issues with **mi** would be to test the sensitivity to model specification. Since **mi** is extremely flexible about the model specification (see Section 4.3.1, when a user is fitting elaborated conditional models, it is probably a good idea to always check the sensitivity of model specification.

Cross Validation

Cross validation of multiple imputation is another thing users can do to check the validity of **mi**. Gelman, King, and Liu (1998) illustrate how to conduct cross validation in a multiple imputation example. We plan to add a cross validation module to **mi** but it is not included in the current version.

4.6. Interactive Interface

mi has an interactive program where users do not have to type commands to perform multiple imputation. By calling `mi.iteractive()` and giving it the data to be imputed, it will walk the users through all the necessary settings and steps as discussed in the previous sections.

```
R> data(CHAIN)
R> IMP <- mi.interactive(CHAIN)

-----------------------------------
creating information matrix:
-----------------------------------

done

-----------------------------------
 Would you like to:
-----------------------------------

1: look at current setting
2: proceed to mi with current setting
3: change current setting


Selection:
```

5. Future Plans

The major goal of **mi** is to make multiple imputation transparent and easy to use for the users. Here are four characteristics of the package that we believe are particularly valuable.

1. Graphical diagnostics of imputation models and convergence of the imputation process.

2. Use of Bayesian versions of regression models to handle the issue of separation.

3. Imputation model specification is similar to the way in which you would fit a regression model in R.

4. Automatical detection of problematic characteristics of data followed by either a fix or an alert to the user. In particular, **mi** adds noise into the imputation process to solve the problem of additive constraints.

As with many other software packages, **mi** is continually being augmented and improved. One caution with the current incarnation is that **mi** may take some time to converge with big datasets with a high rate of missingness across many variables. We are currently investigating approaches to increase the computational efficiency of the algorithm.

Another future direction includes expanding the functionality of **mi** to allow for imputation of time-series cross-sectional data, hierarchical or clustered data. Currently, it is only possible to include group or time indicators as predictors in the imputation process to capture the group-specific or time-specific aspect of missingness patterns. We would like to use multilevel modeling to model these types of data (Gelman and Hill 2007).

Finally, as discussed in Section 4.5, we will incorporate tools and functions to perform sensitivity analysis and cross-validation of **mi**.

Acknowledgements

We thank Peter Messeri for the CHAIN example, Maria Grazia Pittau for helpful discussions and her early works on **mi**, the US Nation Science Foundation, National Institutes of Health, and Institute of Education Sciences for partial support of this research.

References

Abayomi K, Gelman A, Levy M (2008). "Diagnostics for Multivariate Imputations." *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, **57**(3), 273–291.

Bates D, Maechler M (2009). *Matrix: Sparse and Dense Matrix Classes and Methods.* R package version 0.999375-25.

Bates D, Maechler M, Dai B (2008). *lme4: Linear Mixed-Effects Models Using S4 Classes.* R package version 0.999375-28, URL http://lme4.r-forge.r-project.org/.

CHAIN (2009). "NY HIV Data CHAIN." http://www.nyhiv.org/data_chain.html.

Cleveland WS (1979). "Robust Locally Weighted Regression and Smoothing Scatterplots." *Journal of the American Statistical Association*, **74**(368), 829–836.

Fay RE (1996). "Alternative Paradigms for the Analysis of Imputed Survey Data." *Journal of the American Statistical Association*, **91**(434), 490–498.

Fox J (2009). *car: Companion to Applied Regression.* R package version 1.2-14, URL http://www.r-project.org,http://socserv.socsci.mcmaster.ca/jfox/.

Gelman A, Carlin JB, Stern HS, Rubin DB (2004). *Bayesian Data Analysis.* Chapman & Hall/CRC, Boca Raton, Fl., 2nd edition.

Gelman A, Goegebeur Y, Tuerlinckx F, Van Mechelen I (2000). "Diagnostic Checks for Discrete Data Regression Models Using Posterior Predictive Simulations." *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, **49**(2), 247–268.

Gelman A, Hill J (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models.* Cambridge University Press, UK.

Gelman A, Jakulin A, Pittau MG, Su YS (2008). "A Weakly Informative Default Prior Distribution for Logistic and Other Regression Models." *Annals of Applied Statistics*, **2**(4), 1360–1383.

Gelman A, King G, Liu C (1998). "Not Asked and Not Answered: Multiple Imputation for Multiple Surveys." *Journal of the American Statistical Association*, **93**(443), 846–857.

Gelman A, Su YS, Yajima M, Hill J, Pittau MG, Kerman J, Zheng T (2009). *arm: Data Analysis Using Regression and Multilevel/Hierarchical Models.* R package version 1.2-8, URL http://www.stat.columbia.edu/~gelman/software/arm.

Gelman A, Van Mechelen I, Verbeke G, Heitjan DF, Meulders M (2005). "Multiple Imputation for Model Checking: Completed-Data Plots with Missing and Latent Data." *Biometrics*, **61**(1), 74–85.

Little RJA, Rubin DB (2002). *Statistical Analysis with Missing Data.* Wiley, Hoboken, N.J., 2nd edition.

Meng XL (1994). "Multiple-Imputation Inferences with Uncongenial Sources of Input." *Statistical Science*, **9**(4), 538–558.

Messeri P, Lee G, Abramson DM, Aidala A, Chiasson MA, Jessop DJ (2003). "Antiretroviral Therapy and Declining AIDS Mortality in New York City." *Medical Care*, **41**(4), 512–521.

Plate T, Heiberger R (2004). *abind: Combine Multi-Dimensional Arrays*. R package version 1.1-0.

Plummer M, Best N, Cowles K, Vines K (2009). *coda: Output Analysis and Diagnostics for MCMC*. R package version 0.13-4.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org`.

Raghunathan TE, Lepkowski JM, Van Hoewyk J, Solenberger P (2001). "A Multivariate Technique for Multiply Imputing Missing Values Using a Sequence of Regression Models." *Survey Methodology*, **27**(1), 85–95.

Robins JM, Wang N (2000). "Inference for imputation estimators." *Biometrika*, **87**(1), 113–124.

Rubin DB (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley, New York.

Sturtz S, Ligges U, Gelman A (2005). "R2WinBUGS: A Package for Running WinBUGS from R." *Journal of Statistical Software*, **12**(3), 1–16. URL `http://www.jstatsoft.org`.

van Buuren S, Oudshoorn CGM (2000). "Multivariate Imputation by Chained Equations: MICE V1.0 User's Manual." *TNO Report PG/VGZ/00.038*, TNO Preventie en Gezondheid, Leiden. `http://www.stefvanbuuren.nl/publications/MICE%20V1.0%20Manual%20TNO00038%202000.pdf`.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0, URL `http://www.stats.ox.ac.uk/pub/MASS4`.

Affiliation:

Yu-Sung Su
Department of Statistics
Columbia University
1255 Amsterdam Avenue
New York, NY 10027, USA
Department of Humanities and Social Sciences
Steinhardt School of Culture, Education and Human Development
New York University
246 Greene Street, 3rd Floor
New York, NY 10003, USA
E-mail: `yusung@stat.columbia.edu`
URL: `http://www.stat.columbia.edu/~yusung`

Andrew Gelman
Department of Statistics
Columbia University
1255 Amsterdam Avenue
New York, NY 10027, USA
E-mail: gelman@stat.columbia.edu
URL: http://www.stat.columbia.edu/~gelman

Jennifer Hill
Department of Humanities and Social Sciences
Steinhardt School of Culture, Education and Human Development
New York University
726 Broadway, Room 754
New York, NY 10003, USA
E-mail: jennifer.hill@nyu.edu

Masanao Yajima
Department of Statistics
University of California at Los Angeles
8208 Math Science Bldg
Los Angeles, CA 90095-1554, USA
E-mail: yajima@ucla.edu
URL: http://www.stat.ucla.edu/~yajima

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd