Journal of Modern Applied Statistical Methods

Volume 16 | Issue 1 Article 35

5-1-2017

JMASM44: Implementing Multiple Ratio Imputation by the EMB Algorithm (R)

Masayoshi Takahashi
Tokyo University of Foreign Studies, mtakahashi@tufs.ac.jp

Recommended Citation
Takahashi, M. (2017). JMASM44: Implementing multiple ratio imputation by the EMB algorithm (R). Journal of Modern Applied Statistical Methods, 16(1), 657-673. doi: 10.22237/jmasm/1493598900

JMASM44: Implementing Multiple Ratio Imputation by the EMB Algorithm (R)

Cover Page Footnote
The author wishes to thank Mr. Yutaka Abe (Hitotsubashi University) for his valuable comments on an earlier version of this article. However, any remaining errors are the author's responsibility. Also, note that the views and opinions expressed in this article are the author's own, not necessarily those of the institution. The analyses in this article were conducted using R 3.1.0.

Journal of Modern Applied Statistical Methods
May 2017, Vol. 16, No. 1, 657-673.
doi: 10.22237/jmasm/1493598900

ISSN 1538 − 9472

JMASM44: Implementing Multiple Ratio Imputation by the EMB Algorithm (R)

Masayoshi Takahashi
Tokyo University of Foreign Studies
Tokyo, Japan

Although single ratio imputation is often used to deal with missing values in practice, there is a paucity of discussion regarding multiple ratio imputation. Code in the *R* statistical environment is presented to execute multiple ratio imputation by the Expectation-Maximization with Bootstrapping (EMB) algorithm.

Keywords: Multiple imputation, ratio imputation, Expectation-Maximization, bootstrap, missing data, incomplete data, nonresponse, estimation uncertainty

Introduction

Code is presented for multiple ratio imputation step by step in the imputation stage, followed by the analysis stage. The Appendix combines these *R*-codes to present Software `MrImputation` as a collection of R-functions `mrimpute` and `mranalyze`. *R*-function `mrimpute` performs multiple ratio imputation. *R*-function `mranalyze` allows us to conduct statistical analyses using the multiply-imputed data by *R*-function `mrimpute`. Takahashi (2017) offers a detailed explanation. As for single ratio imputation and multiple imputation, see de Waal et al. (2011), Hu et al. (2001), King et al. (2001), Carpenter & Kenward (2013), Honaker & King (2010), Honaker et al. (2011), Little & Rubin (2002).

Preparation Stage

As an illustration, consider the dataset `data`. In the code presented in Appendix, the name of `data` can be defined by option `data=`. Thus, it can be named any way an imputer wants it to be. This small example dataset contains two variables and five units as displayed in Figure 1. The observation for unit 1 in $y1$ is missing

Masayoshi Takahashi is an Assistant Professor of Institutional Research. Email at mtakahashi@tufs.ac.jp.

(NA). Thus, *y*1 is the target incomplete variable for imputation, and *y*2 is the auxiliary complete variable. Also, *y*1 is stored in `data[,1]` and *y*2 in `data[,2]`. This article will use this small dataset for illustration. As this dataset implies, the target variable for imputation needs to be stored in the first column of data, i.e., `data[,1]`, in order to execute the code shown in this article.

```
data<-read.csv("data.csv",header=T)
attach(data)
```

```
> data
        y1        y2
1       NA 10.545612
2 5.779933  9.728869
3 4.835343  9.920130
4 6.219675  8.897375
5 7.012357 10.417368
```

Figure 1. Example of Incomplete Data

The number of multiply-imputed data is set by `M`, where `M > 1`. In this example, it is set to 2 so that the outputs can be visually presented below. To allow reproducibility, the random number seed value needs to be set by function `set.seed`. This step is necessary, because multiple imputation relies on pseudo-random numbers; thus, without setting a seed, there will be no way of reproducing the same results.

```
M<-2
set.seed(1223)
```

Many types of data are skewed to the right in the distribution, i.e., the distribution is not multivariate normal, but multivariate log-normal. If this is the case, a sensible option to deal with such a variable is to use log-transformation, and the imputed values will be unlogged after imputations are completed (Allison, 2002, p.39; Honaker et al., 2011, p.15). In the complete code shown in Appendix, if `log=TRUE`, then the following code log-transforms the data. The default setting is that `log=FALSE`. Obviously, if data are multivariate normal to begin with, this option should be set to `FALSE`.

```
if(log){
  data<-log(data)
}
```

Imputation Stage

Nonparametric Bootstrap

The first step to perform multiple ratio imputation is to implement random draws of $\boldsymbol{\mu}$ from an appropriate posterior distribution to account for estimation uncertainty. The EMB algorithm substitutes the complex process of drawing $\boldsymbol{\mu}$ from the posterior distribution with a nonparametric bootstrapping algorithm, which is a resampling method, where the observed sample is used as the pseudo-population. In other words, a resample of size n is randomly drawn from this observed sample of size n with replacement, and this process is repeated M times (Shao & Tu, 1995; Horowitz, 2001).

R-function `sample(x,size,replace=TRUE)` can be used for this purpose, where `x` is a vector from which to sample, `size` is the number of items to sample, and `replace=TRUE` specifies sampling with replacement. Unfortunately, this function randomly draws a vector, not a matrix. In the process of imputation, the imputer must keep a pair of observations for the two variables. Thus, our code first creates `sampleframe` to randomly draw the row number of data, which is an `nrow(data)` by `M` matrix, where `nrow(data)` is the number of rows in `data`.

```
sampleframe<-matrix(sample(nrow(data),nrow(data)*M,replace=TRUE),
nrow=nrow(data),ncol=M)
```

The resulting matrix obtained from the above code is displayed in Figure 2, where each column contains a vector of the row numbers randomly drawn from the original data. For example, `sampleframe[1,1]` is 4, meaning that this cell refers to row number 4 in the original data, i.e., $y1 = 6.219675$ and $y2 = 8.897375$, `sampleframe[2,1]` is 1, meaning that this cell refers to row number 1 in the original data, i.e., $y1 =$ NA and $y2 = 10.545612$, and so on.

Based on `sampleframe`, our code makes a random draw of the values of $y1$ and $y2$ from the original data M times. First, let us create a list named `datasub` with the elements of NA and then replace these NAs by appropriate values in the original data, so that `datasub[[i]]` obtains `data[sampleframe[,i],]`, and the `for`

loop repeats this process *M* times. In order to use this `datasub` in the EM algorithm below, `datasub` is transformed to a matrix.

```
> sampleframe
     [,1] [,2]
[1,]    4    5
[2,]    1    1
[3,]    2    4
[4,]    2    5
[5,]    1    1
```

Figure 2. Randomly-Drawn Row Numbers

```
datasub<-as.list(rep(NA,M))
for(i in 1:M){
  datasub[[i]]<-as.matrix(data[sampleframe[,i],])
}
```

The resulting bootstrap resamples are shown in Figure 3, where `datasub[[1]]` and `datasub[[2]]` represent the m^{th} bootstrap resample, respectively.

```
> datasub
[[1]]
          y1        y2
4   6.219675  8.897375
1         NA 10.545612
2   5.779933  9.728869
2.1 5.779933  9.728869
1.1       NA 10.545612

[[2]]
          y1        y2
5   7.012357 10.417368
1         NA 10.545612
4   6.219675  8.897375
5.1 7.012357 10.417368
1.1       NA 10.545612
```

Figure 3. Example of Bootstrap Resamples (*M* = 2)

EM Algorithm

Each bootstrap resample created above is likely to be incomplete. Estimates using these resamples are expected to be biased and inefficient. In order to avoid this problem, the EM algorithm is used to refine the estimates in bootstrap resamples. As for the EM algorithm, see Little & Rubin (2002), Do & Batzoglou (2008), the *R*-package `Norm` by Schafer (1997), and the function `em.norm` (Fox, 2015). The current code does not use `Norm` for the sake of generating multiple imputation, but function `em.norm` is useful for the computational purpose of the EM algorithm. First, use the require function to load `Norm` in *R*. In the code below, `p` is the number of columns (variables) in the data, `para` is the number of parameters to be estimated, `thetahat` is an empty matrix with the dimension of `M` by `para`, and `emmu` is an empty matrix with the dimension of `M` by `p`. These are housekeeping issues to perform the EM algorithm by way of function `em.norm`.

```
require(norm)
p<-ncol(data)
para<-p*(p+3)/2+1
thetahat<-matrix(NA,M,para)
emmu<-matrix(NA,M,p)
```

Function `prelim.norm` takes care of the preliminary manipulations for a matrix of incomplete data, which is a necessary step for using `em.norm`, whose results are stored in `thetahat`. Option `showits=FALSE` quietly runs `em.norm`. If the imputer wants to monitor the iteration process of EM, then this option should be set to `TRUE`. Option `maxits=1000` sets the maximum number of iterations to 1,000. Function `getparam.norm` produces the estimated values of the MLEs, which is stored in `emmu`. Option `corr=FALSE` computes the means and variance-covariance matrix. The `for` loop repeats the `em.norm` function to be applied to `datasub` *M* times. This process is the essential part of the EMB algorithm, meaning that the EM algorithm is applied to each of the *M* bootstrap resamples.

```
for(i in 1:M){
  thetahat[i,]<-em.norm(prelim.norm(datasub[[i]]),
  showits=FALSE,maxits=1000)
  emmu[i,]<-getparam.norm(prelim.norm(datasub[[i]]),
  thetahat[i,],corr=FALSE)$mu
}
```

All of the estimates of the means by the EM algorithm are stored in `emmu`. Thus, typing `emmu` returns the following matrix in Figure 4, where the first column refers to the means for the first variable in the data, and the second column refers to the means for the second variable in the data. Also, the first row refers to the means in $m = 1$ and the second row refers to the means in $m = 2$. Note that these are the MLEs of the means.

```
> emmu
         [,1]      [,2]
[1,] 5.695139  9.889267
[2,] 6.880546 10.164667
```

Figure 4. MLEs for the Means of *y*1 and *y*2

Implementation of Multiple Ratio Imputation

Using matrix `emmu` allows us to estimate multiple ratios of two variables as follows. The estimated ratios are stored in `beta`, which is an empty matrix with the dimension of `M` by `ncol(data)-1`. Ratio imputation has only two variables; thus, the number of columns in the data, i.e., `ncol(data)`, is 2, which means that `beta` is essentially an `M` by 1 column vector.

```
beta<-matrix(NA,M,ncol(data)-1)
beta<-emmu[,1]/emmu[,2]
```

Typing `beta` returns a vector of M values, where the first value is the ratio in the first model, the second value in the second model, and so on. This is $\tilde{\beta}$ in equation (6) of Takahashi (2017).

```
> beta
[1] 0.5758909 0.6769082
```

Figure 5. The Values of the Slopes in the Multiple Ratio Imputation Model

As a preparation for multiple ratio imputation, let us define the following matrices. These are housekeeping issues to perform multiple ratio imputation. All of the matrices are empty matrices with the dimensions of `nrow(data)` by `M`.

```
imp<-matrix(NA,nrow(data),M)
resid<-matrix(NA,nrow(data),M)
e<-matrix(NA,nrow(data),M)
imp1<-matrix(NA,nrow(data),M)
imp2<-matrix(NA,nrow(data),M)
```

The values of `beta` are multiplied by `data[,2]` which is the values of the second variable in the data. Specifically, `data[,2]` is *y*2 in our example. Thus, the following code is $\tilde{\beta}Y_{i2}$ in equation (6) of Takahashi (2017). The `for` loop repeats this process *M* times. The imputed values are stored in `imp`, where `imp[,1]` is the imputed data from `m = 1` and `imp[,2]` is the imputed data from `m = 2`.

```
for(i in 1:M){
  imp[,i]<-beta[i]*data[,2]
}
```

To complete the process, a small disturbance term needs to be added to the imputed values, which is $\tilde{\varepsilon}_i$ in equation (6) of Takahashi (2017). In the following code, `resid` is the differences (residuals) between observed values and predicted values. Also, $\tilde{\varepsilon}_i$ is `e[,i]`, which is normally distributed with the mean of 0 and the standard deviation of the residuals, `resid[,i]`. In the last line, `e[,i]` is added to `imp[,i]`. The `for` loop repeats this whole process *M* times.

```
for(i in 1:M){
  resid[,i]<-data[,1]-imp[,i]
  e[,i]<-rnorm(nrow(data),0,sd(resid[,i],na.rm=TRUE))
  imp1[,i]<-imp[,i]+e[,i]
}
```

All of the values were imputed, both observed and missing. What actually needs to be imputed is the missing part of the data only. Therefore, the final step is to replace `NA` with `imp1` and to keep the observed value as is. In the following code, `imp2` is essentially $\tilde{Y}_{i1}$ in equation (6) of Takahashi (2017). If `data[j,1]` is

missing, then `imp2[j,i]` obtains the imputed value `imp1[j,i]`; otherwise, `imp2[j,i]` obtains `data[j,1]`. In the following loop, `i` refers to the number of imputations and `j` refers to the row number in the data.

```
for(i in 1:M){
  for(j in 1:nrow(data)){
    if (is.na(data[j,1])=="TRUE"){
    imp2[j,i]<-imp1[j,i]
  }else{
    imp2[j,i]<-data[j,1]}
}}
```

Remember that log-normal data were log-transformed above. Imputed values must be put back to the original scale of incomplete data. The following code unlogs the log-transformed variables.

```
if(log){
  imp2<-exp(imp2)
  data<-exp(data)
}
```

Some variables have logical bounds. For instance, economic variables such as turnover cannot be negative. If this is the case, `zero=TRUE` can be specified in the complete code in Appendix. This option forces negative imputed values to be zero. Warning is that this option may suppress the correct uncertainty in the imputation model (Honaker et al., 2011, pp. 23-25); thus, this option should be used cautiously. The default setting is `zero=FALSE`.

```
if(zero){
  imp2[which(imp2<0)]<-0
}
```

Finally, `imp2` returns the following two sets of imputed data, because $M = 2$. The values in row [1,] change over columns [,1] to [,2], because these values are imputed values. The values in the other rows do no change over columns, because these are observed values.

```
> imp2
         [,1]     [,2]
[1,] 6.739130 6.828206
```

```
[2,] 5.779933 5.779933
[3,] 4.835343 4.835343
[4,] 6.219675 6.219675
[5,] 7.012357 7.012357
```

Figure 6. Example of Multiply-Imputed Data

The `write.csv` function saves the imputed data along with the original data as follows, where `y1` is the original incomplete variable, `y2` is the original auxiliary variable, and `imp2` is a matrix of *M* imputed data created above.

```
y1<-data[,1]; y2<-data[,2]
impdata<-data.frame(y1,y2,imp2)
write.csv(impdata,"mridata.csv",row.names=FALSE)
```

Figure 7 contains the output data named `mridata` in the csv format, which can be reloaded in *R* or any statistical software of an analyst's choice for subsequent statistical analyses. In this output dataset, Column A (*y*1) is the original incomplete data, Column B (*y*2) is the original auxiliary variable, and Columns C to D (*X*1, *X*2) are the multiply imputed data.

A10

	A	B	C	D
1	y1	y2	X1	X2
2	NA	10.54561	6.73913	6.828206
3	5.779933	9.728869	5.779933	5.779933
4	4.835343	9.92013	4.835343	4.835343
5	6.219675	8.897375	6.219675	6.219675
6	7.012357	10.41737	7.012357	7.012357

Figure 7. Example of Output Data (csv file)

Analysis Stage

Mean and Standard Deviation

After reading `mridata.csv`, various statistical analyses can be performed. To calculate the mean and the standard deviation of an imputed variable ($y1$), the analyst first creates two empty vectors of `means` and `sds`, and repeats the calculations M times by the `for` loop. Typing `means` and `sds` returns M values of the means and the standard deviations.

```
means<-c(NA); sds<-c(NA)
  for(k in 1:M){
  means[k]<-mean(imp2[,k])
  sds[k]<-sd(imp2[,k])
}
```

To calculate a combined point estimate, the analyst simply takes the average by equation (7) of Takahashi (2017). Furthermore, by calculating the standard deviation of `means`, i.e. `sd(means)`, the analyst can estimate the amount of estimation uncertainty due to imputation as a confidence interval.

```
mean(means)                 #Combined Point Estimate of Mean
mean(sds)                   #Combined Point Estimate of Std. Dev.
sd(means)                   #Estimation Uncertainty
mean(means)+2*sd(means)     #Confidence Interval Upper Limit
mean(means)-2*sd(means)     #Confidence Interval Lower Limit
```

Consider again the example data in Figure 1. The combined point estimate of the means is 6.126, with the combined point estimate of standard deviation 0.868. Estimation uncertainty is measured by `sd(means)`, which is the standard deviation of the M means, or the standard error of the estimated M means. In our case, it is 0.013. Therefore, there is an approximately 95% confidence that the true mean of complete data is somewhere between 6.101 and 6.151, after taking the error due to missingness into account.

Regression of $y2$ on $y1$

Suppose that $y2$ is the dependent variable and $y1$ is the explanatory variable in regression. To estimate the regression coefficients and the associated standard

errors, the analyst first creates four empty vectors, `reg1`, `reg2`, `reg3`, and `reg4`. The `for` loop repeats the estimation of regression models *M* times. The results are stored in `summary(model)$coefficients[i]`, where *i* = 1 and 3 are regression coefficients and *i* = 2 and 4 are standard errors.

```
reg1<-c(NA); reg2<-c(NA); reg3<-c(NA); reg4<-c(NA)
for(k in 1:M){
  model<-lm(data[,2]~data[,k+2])
  reg1[k]<-summary(model)$coefficients[1]
  reg2[k]<-summary(model)$coefficients[2]
  reg3[k]<-summary(model)$coefficients[3]
  reg4[k]<-summary(model)$coefficients[4]
}
```

After the analysis stage is complete, there are *M* values of outputs. Using equations (7) and (8) of Takahashi (2017), the results are combined as follows.

```
intercept<-mean(reg1)              #Combined Intercept
WV1<-mean(reg3^2)                  #Within-Imputation Variance
BV1<-sum((reg1-intercept)^2)/(M-1) #Between-Imputation Variance
TV1<-WV1+(1+1/(M))*BV1             #Total Variance
TSE1<-sqrt(TV1)                    #Total Std. Error
tstat1<-intercept/TSE1             #t-statistics for Intercept
slope<-mean(reg2)                  #Combined Slope
WV2<-mean(reg4^2)                  #Within-Imputation Variance
BV2<-sum((reg2-slope)^2)/(M-1)     #Between-Imputation Variance
TV2<-WV2+(1+1/(M))*BV2             #Total Variance
TSE2<-sqrt(TV2)                    #Total Std. Error
tstat2<-slope/TSE2                 #t-statistics for Slope
```

Consider again the example data in Figure 1. The combined point estimate of the regression intercept is 8.231, with the total standard error of 2.512. Thus, the *t*-statistic for the intercept is 3.277. The combined point estimate of the regression slopes is 0.273 with the total standard error of 0.407. Thus, the *t*-statistic for the slope is 0.671.

Conclusion

It was outlined here how to implement multiple ratio imputation in *R*, which can be easily copied and pasted into *R* for use (See Appendix). These codes estimate multiple ratio imputation, and statistically analyze imputed data by multiple ratio imputation. Therefore, this will be a valuable addition to the choice for imputation techniques.

However, the code described here is only a first step toward implementing multiple ratio imputation; thus, the code is expected to be updated so as to maximize computational efficiency and to expand the scope of data that can be handled. Furthermore, the EMB algorithm is a general approach composed of the EM algorithm and nonparametric bootstrapping. Therefore, multiple ratio imputation can be implemented not only in *R*, but also in other statistical environments. Also, multiple ratio imputation is not limited to the EMB algorithm. Depending on the nature of imputation, multiple ratio imputation may be implemented by way of other multiple imputation algorithms, such as MCMC and Fully Conditional Specification (FCS) (van Buuren, 2012).

Acknowledgments

The author wishes to thank Mr. Yutaka Abe (Hitotsubashi University) for his valuable comments on an earlier version of this article. The author also wishes to thank the two anonymous reviewers for reviewing this article. However, any remaining errors are the author's responsibility. Also, note that the views and opinions expressed in this article are the author's own, not necessarily those of the institution. The analyses in this article were conducted using *R*.3.1.0.

References

Allison, P. D. (2002). *Missing Data*. Thousand Oaks, CA: Sage Publications.

Carpenter, J. R., & Kenward, M. G. (2013). *Multiple Imputation and its Application*. Chichester, West Sussex: John Wiley & Sons. doi: 10.1002/9781119942283

de Waal, T., Pannekoek, J., & Scholtus, S. (2011). *Handbook of Statistical Data Editing and Imputation*. Hoboken, NJ: John Wiley & Sons. doi: 10.1002/9780470904848

Do, C. B., & Batzoglou, S. (2008). What is the expectation maximization algorithm? *Nature Biotechnology, 26*(8), 897-899. doi: 10.1038/nbt1406

Fox, J. (2015). Package 'Norm' [Computer software]. Retrieved from: http://cran.r-project.org/web/packages/norm/norm.pdf

Honaker, J., & King, G. (2010). What to do about missing values in time series cross-section data. *American Journal of Political Science, 54*(2), 561-581. doi: 10.1111/j.1540-5907.2010.00447.x

Honaker, J., King, G., & Blackwell, M. (2011). Amelia II: a program for missing data. *Journal of Statistical Software, 45*(7), 1-47. doi: 10.18637/jss.v045.i07

Horowitz, J. L. (2001). The bootstrap. In J. J. Heckman & E. Leamer (Eds), *Handbook of Econometrics* (pp. 3160-3228), Vol. 5. Amsterdam: Elsevier. doi: 10.1016/s1573-4412(01)05005-x

Hu, M., Salvucci, S., & Lee, R. (2001). *A Study of Imputation Algorithms. Working Paper No. 2001–17*. U.S. Department of Education. National Center for Education Statistics. Retrieved from: http://nces.ed.gov/pubs2001/200117.pdf

King, G., Honaker, J., Joseph, A., & Scheve, K. (2001). Analyzing incomplete political science data: an alternative algorithm for multiple imputation. *American Political Science Review, 95*(1), 49-69.

Little, R. J. A., & Rubin, D. B. (2002). *Statistical Analysis with Missing Data* (2nd ed). Hoboken, NJ: John Wiley & Sons. doi: 10.1002/9781119013563

Schafer, J. L. (1997). *Analysis of Incomplete Multivariate Data*. Boca Raton, FL: Chapman & Hall/CRC.

Shao, J., & Tu, D. (1995). *The Jackknife and Bootstrap*. New York, NY: Springer. doi: 10.1007/978-1-4612-0795-5

Takahashi, M. (2017). Multiple ratio imputation by the EMB algorithm: Theory and simulation. *Journal of Modern Applied Statistical Methods*, 16(1). doi: 10.22237/jmasm/1493598840

van Buuren, S. (2012). *Flexible Imputation of Missing Data*. Boca Raton, FL: Chapman & Hall/CRC. doi: 10.1201/b11826

Appendix: Software `MrImputation`

Software `MrImputation` (version 1.0.0), which stands for multiple ratio imputation, is a collection of *R*-functions explained step by step in this article. This appendix combines each of the steps as a set of *R*-functions `mrimpute` and `mranalyze`.

User Manual

Copy the following codes into the *R* script and save them as `mrimpute.R` and `mranalyze.R` on the computer. After reading an appropriate data file in *R*, use function source to read these functions as follows.

```
source("mrimpute.R")
source("mranalyze.R")
```

Description of `mrimpute` This function performs the imputation stage of multiple ratio imputation and produces multiply-imputed data named mridata.csv.

Usage `mrimpute(data = data, M = 100, seed = 1223, log = FALSE, zero = FALSE, outdata = TRUE)`

Arguments

`data`	A data frame that contains the incomplete variable targeted for imputation. The imputer can specify any name of the data to be used.
`M`	The number of multiply-imputed datasets. The imputer can set any number.
`seed`	Random number seed value. Any number can be specified.
`log`	An option to log-transform the data. The default is FALSE. If log-transformation is optimal, then this option should be set to TRUE.
`zero`	An option to suppress negative values to zero. The default is FALSE. If negative imputed values are unacceptable, this option should be set to TRUE.
`outdata`	An option to save the imputed data as a csv file. The default is TRUE.

Description of `mranalyze` This function performs the analysis stage. It returns the mean and the standard deviation of the imputed variable. It can also return the result of regression analysis of $y2$ on $y1$ if `reg=TRUE`.

Usage `mranalyze(data, reg = FALSE)`

Arguments

`data`	The mridata.csv created by mrimpute.
`reg`	An option to perform regression analysis. The default is FALSE. If the analyst wants to see the result of regression analysis, this option should be set to TRUE.

R-Function `mrimpute`: Imputation Stage

```
mrimpute<-function(data,M,seed,outdata=TRUE,log=FALSE,zero=FALSE){
data<-data; M<-M; seed<-seed; set.seed(seed)
if(log){data<-log(data)}
sampleframe<-matrix(sample(nrow(data),nrow(data)*M,
             replace=TRUE),nrow=nrow(data),ncol=M)
datasub<-as.list(rep(NA,M))
for(i in 1:M){datasub[[i]]<-as.matrix(data[sampleframe[,i],])}
suppressMessages(suppressWarnings(require(norm)))
p<-ncol(data); para<-p*(p+3)/2+1; thetahat<-matrix(NA,M,para)
emmu<-matrix(NA,M,p)
for(i in 1:M){thetahat[i,]<-em.norm(prelim.norm(datasub[[i]]),
                           showits=FALSE,maxits=1000)
              emmu[i,]<-getparam.norm(prelim.norm(datasub[[i]]),
                           thetahat[i,],corr=FALSE)$mu}
imp0<-as.list(rep(NA,M)); imp<-matrix(NA,nrow(data),M)
resid<-matrix(NA,nrow(data),M); e<-matrix(NA,nrow(data),M)
imp1<-matrix(NA,nrow(data),M); beta<-matrix(NA,M,ncol(data)-1)
beta<-emmu[,1]/emmu[,2]
for(i in 1:M){imp[,i]<-beta[i]*data[,2]}
for(i in 1:M){resid[,i]<-data[,1]-imp[,i]
              e[,i]<-rnorm(nrow(data),0,sd(resid[,i],na.rm=TRUE))
              imp1[,i]<-imp[,i]+e[,i]}
imp2<-matrix(NA,nrow(data),M)
for(i in 1:M){imp2[,i]<-data[,1]}
for(i in 1:M){
  for(j in 1:nrow(data)){
    if (is.na(data[j,1])=="TRUE"){
```

```
    imp2[j,i]<-imp1[j,i]
    }else{
    imp2[j,i]<-data[j,1]}
  }}
if(log){imp2<-exp(imp2);data<-exp(data)}
if(zero){imp2[which(imp2<0)]<-0}
impdata<-data.frame(data, imp2)
  if (outdata){
    write.csv(impdata,"mridata.csv",row.names=FALSE)
  }
}
```

R-Function `mranalyze`: Analysis Stage

```
mranalyze<-function(data,reg=FALSE){
data<-data; M<-ncol(data)-2; means<-c(NA); sds<-c(NA)

for(k in 1:M){
  means[k]<-mean(data[,k+2])
  sds[k]<-sd(data[,k+2])
}
meanimp<-mean(means);BISD<-sd(means);UL<-mean(means)+2*sd(means);LL<-
           mean(means)-2*sd(means);sd<-mean(sds)
outmatrix1<-matrix(c(meanimp, sd, BISD, UL, LL))
colnames(outmatrix1)<-"Summary"
rownames(outmatrix1)<-c("mean","sd","BISD","95%CIUL","95%CILL")

if(reg){
reg1<-c(NA); reg2<-c(NA); reg3<-c(NA); reg4<-c(NA)
for(k in 1:M){
  model<-lm(data[,2]~data[,k+2])
  reg1[k]<-summary(model)$coefficients[1]
  reg2[k]<-summary(model)$coefficients[2]
  reg3[k]<-summary(model)$coefficients[3]
  reg4[k]<-summary(model)$coefficients[4]
}

intercept<-mean(reg1)
```

```
WV1<-mean(reg3^2)
BV1<-sum((reg1-intercept)^2)/(M-1)
TV1<-WV1+(1+1/(M))*BV1
TSE1<-sqrt(TV1)
tstat1<-intercept/TSE1

slope<-mean(reg2)
WV2<-mean(reg4^2)
BV2<-sum((reg2-slope)^2)/(M-1)
TV2<-WV2+(1+1/(M))*BV2
TSE2<-sqrt(TV2)
tstat2<-slope/TSE2

outmatrix2<-matrix(c(intercept, TSE1, tstat1, slope, TSE2, tstat2))
colnames(outmatrix2)<-"Regression"
rownames(outmatrix2)<-c("intercept","TSE(intercept)","t-
           Stat(intercept)","slope","TSE(slope)" ,"t-Stat(slope)")
}

if(reg){
result<-list(outmatrix1, outmatrix2)
  return(result)
}else{
result<-list(outmatrix1)
  return(result)
}
}
```