imputeMulti: Imputation for Multivariate Multinomial Missing Data

Alex Whitworth

Abstract

imputeMulti is an R package for imputation of multivariate multinomial missing data via expectation-maximization and data augmentation algorithms. The package allows the specification of Bayesian priors, including data-dependent priors. For performance, calculation of the summary statistics of the multinomial distribution is implemented in C++; **imputeMulti** also supports these calculations in parallel. As a result, the **imputeMulti** package capably handles large datasets. In this article, we introduce the package's functionality and provide a hands-on approach to solving multinomial missing data problems.

Keywords: multinomial, missing data, imputation, expectation-maximization, data-augmentation, EM, DA, R.

1. Introduction

As Honaker, King, and Blackwell (2011) noted, "missing data is a ubiquitous problem [especially] in social science data. Respondents do not answer every question, countries do not collect statistics every year, archives are incomplete, [and] subjects drop out of panels. Most statistical analysis methods, however, assume the absence of missing data, and are only able to include observations for which every variable is measured." In the past several decades, widely available methods have been developed for missing value imputation allowing practitioners to move beyond ad-hoc methods such as casewise-deletion and mean imputation, to more advanced methods such as multiple imputation Rubin (1987), expectation maximization Dempster, Laird, and Rubin (1977), chained equations van Buuren and Groothuis-Oudshoorn (2011), and data augmentation Tanner and Wong (1987). However, many of these methods assume the data comes from a multivariate normal distribution, which ignores missing data from a variety of other distributions.

imputeMulti performs missing data imputation for the common multivariate multinomial distribution. Like other imputation methods, this method creates a "filled in" version of the incomplete data so that analyses which require complete observations can appropriately use all the data in a dataset containing missingness. **imputeMulti** uses the familiar expectation-maximization (EM) and data augmentation (DA) algorithms to estimate the parameter estimates of the multinomial distribution and maximum likelihood to impute missing observations.

Two issues which make multivariate multinomial missing data imputation challenging are the memory required to store all possible combinations of the variables of interest and the computation time required to calculate both the complete data sufficient statistics and the missing

data marginal sufficient statistics. To overcome these performance challenges, **imputeMulti** uses C++ via **Rcpp** Eddelbuettel and Francois (2011) and also allows for parallel processing via **parallel** R Core Team (2016).

imputeMulti provides advantages over other popular imputation methods such as **Amelia** Honaker *et al.* (2011), **mice** van Buuren and Groothuis-Oudshoorn (2011), and **Hmisc** Harrell Jr and others. (2016) when working with multinomial data. The most important advantage is higher imputaton accuracy for multinomial data relative to any of **Amelia**, **mice**, and **Hmisc**. Secondly, **imputeMulti** provides researchers easy access to the multinomial parameter estimates. In many cases, the parameter estimates may be of more interest to researchers than the observation level imputations. One disadvantage of **imputeMulti** is longer run time than **Amelia** or **Hmisc**, although **imputeMulti** is faster than **mice**. Run time is impacted by unique aspects of the the multinomial distribution which will be discussed below.

The remainder of the paper proceeds as follows: Section 2 provides a brief discussion of multinomial missing data problems; Section 3 provides a user's guide to **imputeMulti**; Section 4 compares **imputeMulti** with alternative imputation methods; and Section 5 concludes.

2. Multinomial missing data

imputeMulti is an implementation of the imputation methods for multivariate multinomial missing data found in (Schafer 1997, Chapter 7). An abbreviated discussion of multivariate multinomial missing data is provided below. To facilitate use of this reference text, the same notation as Schafer (1997) is used throughout this discussion.

To begin, let $Y_1, Y_2, \ldots, Y_p$ be categorical variables each taking a finite number of values $Y_j \in \{1, 2, \ldots, d_j\}; j = 1, 2, \ldots, p$. If the observations are independent and identically distributed (iid), then we can reduce Y to a contingency table with D cells, where $D = \prod_{j=1}^{p} d_j$ is the number of distinct combinations of the levels of $Y_1, Y_2, \ldots, Y_p$. The cell counts of D are denoted by $x_d, d = 1, \ldots, D$. If the sample size is n, then x has a multinomial distribution:

$$x|\theta \sim M(n, \theta) \tag{1}$$

with parameter vector $\theta = (\theta_1, \theta_2, \ldots, \theta_D)$. The likelihood function for the multinomial parameter θ is

$$L(\theta|Y) \propto \prod_{d=1}^{D} \theta_d^{x_d} I_{\Theta(\theta)} \tag{2}$$

where $I_{\Theta(\theta)}$ is an indicator function equal to 1 if $\theta \in \Theta$ and 0 otherwise. This leads to the well known maximum likelihood estimates (MLE):

$$\hat{\theta}_d = \frac{x_d}{n}, d = 1, \ldots, D \tag{3}$$

2.1. The Bayesian case and the Dirichlet prior

The simplest way to conduct Bayesian inference on a multinomial model is to assume a Dirichlet prior Ferguson (1973), which is typically denoted by the parameter vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_D)$. The prior and posterior of θ with a Dirichlet prior are thus written in shorthand as

$$\theta|\alpha \sim D(\alpha) \tag{4}$$
$$\theta|Y \sim D(\alpha') \tag{5}$$

where $\alpha' = (\alpha_1 + x_1, \alpha_2 + x_2, \ldots, \alpha_D + x_D)$. The posterior mean is

$$\mathsf{E}(\theta|Y) = \left(\frac{\alpha'_1}{\alpha'_0}, \frac{\alpha'_2}{\alpha'_0} \ldots, \frac{\alpha'_D}{\alpha'_0}\right) \tag{6}$$

with $\alpha'_0 = \sum_{d=1}^{D}(\alpha_d + x_d) = \alpha_0 + n$.

As is typical in Bayesian analyses, the choice of prior can have important effects on the outcome of missing value imputation for multinomial data. In addition to no prior, **imputeMulti** supports three choices of prior. When little prior information is known, a noninformative prior may be a sensible choice. If the sample size is large relative to the number of parameters being estimated, a noninformative prior will have little impact on model inferences. The choice used in **imputeMulti** is twice the Jefferys prior, $c = 1$, which provides proper posterior modes of θ.[1] Another choice of prior is a flattening prior $\alpha = (c, c, \ldots, c)$ for some $c > 1$. A final choice available in **imputeMulti** is the data-dependent prior, whose aim is to take a *peek* at the data, via some summary statistics for instance, prior to analysis. Discussion and criticism of the use of data-dependent priors is beyond the scope of this article. Interested readers are referred to Darnieder (2011).

2.2. Characterizing the multivariate multinomial missing data problem

At the core of all imputation methods is that we do not completely observe the data x, but instead only observe part of it. In the multinomial case, assume that we have grouped the observed data into their observed patterns x^{obs} and their missingness patterns x^{mis}. Index the missingness paterns by $s = 1, 2, \ldots, S$ and define a set of indicator variables:

$$r_{sj} = \begin{cases} 1 & \text{if } Y_j \text{ is observed in } s \\ 0 & \text{if } Y_j \text{ is missing in } s \end{cases}$$

Let $O_s(y)$ and $M_s(y)$ respectively denote the sets of observed and missing variables within each missingness pattern and with elements defined

$$O_s(y) = \{y_j : r_{sj} = 1\} \tag{7}$$
$$M_s(y) = \{y_j : r_{sj} = 0\}. \tag{8}$$

[1] Using the Jefferys prior, $c = \frac{1}{2}$, can produce improper posterior modes. Specifically, if $\alpha_y < 1$ and the corresponding count x_y is zero, then $\hat{\theta}_y$ will be negative.

Since most missing observations are *partially* observed, within each missingness pattern s, observations are cross-classified by their observed variables. The distinct partially observed patterns are denoted $x^{(s)}$; and their counts tabulated into a table denoted

$$z^s_{O_s(y)} = \sum_{M_s(y) \in M_s} x^{(s)} \text{ for all } O_s(y) \in O_s. \tag{9}$$

The marginal probability that an observation falls within a given cell of this table of partially observed values is denoted

$$\beta_{O_s(y)} = \sum_{M_s(y) \in M_s} \theta_{y\cdot} \ . \tag{10}$$

And the observed-data loglikelihood contribution from the partially observed data is

$$l(\theta | Y_{obs}) = \sum_{s=1}^{S} \sum_{O_s(y) \in O_s} z^s_{O_s(y)} log\left(\beta_{O_s(y)}\right). \tag{11}$$

The EM algorithm for maximizing the complete-data loglikelihood is straightforward, with the multivariate case first described by Fuchs (1982). For the E-step, we find the expected summary statistics given the observed data and the current value of theta,

$$\mathsf{E}(x_y | Y_{obs}, \theta) = \sum_{s=1}^{S} z^s_{O_s(y)} \theta_y / \beta_{O_s(y)}. \tag{12}$$

This leads to the trivial M-step. Under maximum-likelihood, set $\hat{\theta} = \mathsf{E}(x_y | Y_{obs}, \theta)/n$ for all $y \in Y$. A minor modification is made to maximize the complete-data posterior density under a Dirichlet prior $\hat{\theta}_y = (x_y + \alpha_y - 1)/(n + \alpha_0 - D)$ for all $y \in Y$ where $\alpha_0 = \sum_i^D \alpha_i$ and D is the number of parameters. Similarly, different minor modifications can be made to convert the EM algorithm to data-augmentation. For DA, instead of proportionally allocating the counts of partially missing observations to fully observed x^{obs} counts based on the current value of θ as in Equation 12, the proportional allocation is replaced by a random allocation based on the current value of θ.

3. Software user's guide

We now turns to the practical matter of using **imputeMulti**, which is freely available as a package for the statistical software R R Core Team (2016) and can be run in any environment that R can. R is freely available from `https://www.r-project.org/`. To install **imputeMulti** from a CRAN (Comprehensive R Archive Network) mirror, simply type the following command in the R command prompt,

```
R> install.packages("imputeMulti").
```

If you wish to use the most current development version, you can install it from Github. This is most easily done using the **devtools** package Wickham and Chang (2016), via

```
R> devtools::install_github("alexWhitworth/imputeMulti").
```

To keep **imputeMulti** up to date, you should use the R command `update.packages()`.

3.1. Example data

To illustrate **imputeMulti**, we use simulated (ACS) American Community Survey data for individuals living in census tract 2221 in Los Angeles County, California. The data was simulated using spatial microsimulation Orcutt *et al.* (1961). This dataset contains ten variables on 3,974 individuals. Missing values have been inserted, independently and at random, to seven of the ten variables. A detailed description of the dataset can be found by typing `?tract2221` into the R console.

```
R> library("imputeMulti")
R> data("tract2221")
```

Beyond being useful to illustrate the use of **imputeMulti**, the data also serves to illustrate one constraint on multivariate multinomial missing data imputation—combinatorial explosion. If we choose to use all ten variables in the model, the number of distinct combinations which may be observed is extremely high. Using the previously introduced notation, $D = \prod_{j=1}^{p} d_j = 8,467,200$. Given that seven variables have missing values, the number of possible distinct combinations of marginally observed variables is even higher at 38,707,200. In R, an integer vector of this size requires 155 MB of memory and a corresponding `data.frame`, which is needed to store $z^s_{O_s(y)}$, requires 1.7 GB of memory. If our model were substantially larger, we would quickly run into **base** R's constraint on matrix size of 2 billion elements.

One option to alleviate the memory constraint is the use of a package that stores objects locally, such as **bigmemory** Kane, Emerson, and Weston (2013). But alleviating memory concerns does not impact the larger problem of factorial runtime—O(p!)—factorial in the number of variables p. One approach is the use of a multivariate normal distribution to provide an approximate solution. The approach suggested in this guide is to instead find an exact solution to an approximate problem. That is, to decompose the set of all variables into subsets of related variables; and to find an exact solution for each subset.

Run-time constraints are not the only benefit of decomposing an increasingly large set of variables. An additional benefit is that complex, higher-order, interactions may be poorly estimated by the fully saturated multinomial model. In these situations, it is often useful to simplify the model by selectively removing some of the highest order associations.

In this dataset, for example, the researcher might decide that income, poverty and employment statuses, educational attainment, age, and gender are closely related and thus group `age`, `gender`, `edu_attain`, `pov_status`, `emp_status`, and `ind_income` into one subset. Further, she might consider marital status, race, nativity, and geographic mobility to all be closely related. This would lead to a second subset of `marital_status`, `nativity`, `geog_mobility`, and `race`. This is the approach that we uses here to illustrate the use of **imputeMulti**.

3.2. Imputation via expectation-maximization

The primary user-facing function for **imputeMulti** is `multinomial_impute`, which provides observation level imputation for multinomial data. Various options can be specified to perform multinomial imputations using EM or DA along with the specification of one of four possible priors: `'none'`, `'non.informative'`, `'flat.prior'`, or `'data.dep'`. These options for imputing missing values via EM are now illustrated, starting without the use of a Bayesian prior.

```
R> set.seed(2134L)
R> imputeEM_none <- multinomial_impute(tract2221[,c(1:2,4,7,9)],
     method = "EM", conj_prior = "none", verbose = TRUE)
```

Here, `method = "EM"` is specified for the expectation-maximization algorithm and `conj_prior = "none"` for a strictly maximum likelihood estimate. The parameter `verbose = TRUE` is specified to provide informative intermediate outputs. The option `parallel = TRUE` can also be specified to compute the sufficient statistics in parallel. In practice, this only provides a performance improvement for exceptionally large datasets and thus the default is currrently `parallel = FALSE`.

```
[1] "Calculating observed data sufficient statistics."
[1] "Setting up Iteration 1."
Iteration 1 : log-likelihood = -32083.8393739337
 Convergence Criteria = 0.0160612788 ...
Iteration 2 : log-likelihood = -24079.2963400782
 Convergence Criteria = 0.0043943837 ...
.... output truncated ....
Iteration 119 : log-likelihood = -23650.2292737947
 Convergence Criteria = 0.0000004931 ...
[1] "Imputing missing observations via MLE results."
```

Typical of EM, convergence is quite rapid. By changing the argument `conj_prior`, Bayesian priors may be specified. The other options for `conj_prior` are shown below

```
R> imputeEM_non <- multinomial_impute(tract2221[,c(1:2,4,7,9)],
     method = "EM", conj_prior = "non.informative", verbose = TRUE)
R> imputeEM_flat <- multinomial_impute(tract2221[,c(1:2,4,7,9)],
     method = "EM", conj_prior = "flat.prior", alpha = 10L, verbose = TRUE)
R> imputeEM_data <- multinomial_impute(tract2221[,c(1:2,4,7,9)],
     method = "EM", conj_prior = "data.dep", verbose = TRUE)
```

where a flat prior may be specified as a scalar by the parameter `alpha`. Note the use of a strong flat prior in this example.

In addition to observation level imputation, some users may only be interested in the parameter estimates. **imputeMulti** allows for this type of analysis as well. To do so, first compute the observed and marginally-observed summary statistics via `multinomial_stats` and then

estimate the parameters directly with your choice of prior via `multinomial_em`. An example of this is shown below.[2]

```
R> x_y <- multinomial_stats(tract2221[,c(1:2,4,7,9)], output = "x_y")
R> z_Os_y <- multinomial_stats(tract2221[,c(1:2,4,7,9)], output = "z_Os_y")
R> x_possible <- multinomial_stats(tract2221[,c(1:2,4,7,9)],
     output = "possible.obs")

R> imputeEM_mle <- multinomial_em(x_y, z_Os_y, x_possible,
     n_obs = nrow(tract2221), conj_prior = "none", verbose = TRUE)

[1] "Setting up Iteration 1."
Iteration 1 : log-likelihood = -32096.9776607773
 Convergence Criteria = 0.0157098212 ...
Iteration 2 : log-likelihood = -24086.0109006056
 Convergence Criteria = 0.0030262731 ...
 .... output truncated ....
Iteration 101 : log-likelihood = -23650.2320024124
 Convergence Criteria = 0.0000004888 ...
```

The outputs of these functions will be examined in Section 3.4.

3.3. Imputation via data-augmentation

Using **imputeMulti** for DA is very similar to the use for expectation-maximization. The chief functional difference is using `method = "DA"` instead of `method = "EM"`. Imputation of observation level data is still done via `multinomial_impute`; and, the choice of prior can be controlled by `conj_prior`. An example is shown below using the second subset of variables discussed at the beginning of this Section.

```
R> imputeDA_none <- multinomial_impute(tract2221[,c(3,6,9:10)],
     method = "DA", conj_prior = "none", verbose = TRUE, burnin = 100)

[1] "Calculating observed data sufficient statistics."
[1] "Setting up Iteration 1."
Iteration 1 : log-likelihood = -26927.4017334082 ...
Iteration 2 : log-likelihood = -18571.4267260919 ...
 .... output truncated ....
Iteration 99 : log-likelihood = -18369.0208750788 ...
Iteration 100 : log-likelihood = -18381.5414370566 ...
[1] "Imputing missing observations via MLE results."
```

Parameter estimates via DA can also be easily obtained via `multinomial_data_aug` in a similar fashion to EM.

[2]The differences in log-likelihood and convergence criteria in the displayed output of `multinomial_impute` and `multinomial_em` are due to random seeding of initial parameters. These random differences in initial parameter values lead to differences of only (<0.003) in final log-likelihood.

```
R> imputeDA_mle <- multinomial_data_aug(x_y, z_Os_y, x_possible,
     conj_prior = "none", verbose = TRUE,
     burnin = 100, post_draws = 1000)
```

With DA, the number of burnin samples is controled via `burnin` for both `multinomial_impute` and `multinomial_data_aug`. `burnin` defaults to 100. Parameter estimates are calculated as the posterior mean based on `post_draws` draws, which can again be specified for both functions. The default is 1000 draws.

3.4. Examining imputed outputs

imputeMulti uses S4 classes. Both `multinomial_em` and `multinomial_data_aug` return `mod_imputeMulti-class` objects while `multinomial_impute` returns `imputeMulti-class` objects. The `imputeMulti-class` class inherits from the `mod_imputeMulti-class`. Several methods are available for summarization and extracting elements from the class objects. However, most users will be interested in only two methods of the `imputeMulti-class` object: `get_parameters` which extracts the parameter estimates, and `get_imputations` which extracts the imputed observation level data.[3]

```
R> param_est    <- get_parameters(imputeEM_none)
R> imputed_data <- get_imputations(imputeEM_none)
```

Since neither `multinomial_em` nor `multinomial_data_aug` impute at the observation level, the `get_imputations` method does not exist for `mod_imputeMulti-class` objects.

To recombine imputed data with the original dataset, utilize the fact that **imputeMulti** maintains both row order and rownames. Recombining imputed and original data can therefore be done via a simple replacement or by merging via rownames. The latter method is implemented in **imputeMulti** and both methods are shown below:

```
R> tract_imp <- data.frame(imputed_data, tract2221[, c(3,6,9:10)])
R> tract_imp <- merge_imputed(imputeEM_none, tract2221[,c(3,6,9:10)])
```

Researchers can also examine the parameter estimates directly. For example, researchers may be interested in the marginal distribution of $\hat{\theta}$ by gender and educational attainment among those aged eighteen to thirty-four compared to those aged fifty to sixty-four. Here a comparison is shown for a `"non.informative"` and `"flat.prior"`, which also illustrates the impact of the previously chosen strong flat prior.

```
R> param_est_non <- get_parameters(imputeEM_non)
R> param_est_flat <- get_parameters(imputeEM_flat)
```

We first extract the marginal statistics and then normalize $\hat{\theta}$ within each age group. The impact of the choice of prior on $\hat{\theta}$ is then shown in Tables 1 and 2.

[3]Other methods for these classes can be found in the documentation. See `?'mod_imputeMulti-class'` and `?'imputeMulti-class'`.

```
R> pe_non18 <- param_est_non[param_est_non$age %in%
     c("18_24", "25_29", "30_34"),]
R> pe_non50 <- param_est_non[param_est_non$age %in%
     c("50_54", "55_59", "60_64"),]
R> non_theta18_34 <- sum(pe_non18$theta_y)
R> non_theta50_64 <- sum(pe_non50$theta_y)
R> marg_non18 <- tapply(pe_non18$theta_y,
     list(pe_non18$gender, pe_non18$edu_attain), sum)
R> marg_non50 <- tapply(pe_non50$theta_y,
     list(pe_non50$gender, pe_non50$edu_attain), sum)
R> round(marg_non18  / non_theta18_34, 4)
R> round(marg_non50  / non_theta50_64, 4)


R> pe_flat18 <- param_est_flat[param_est_flat$age %in%
     c("18_24", "25_29", "30_34"),]
R> pe_flat50 <- param_est_flat[param_est_flat$age %in%
     c("50_54", "55_59", "60_64"),]
R> flat_theta18_34 <- sum(pe_flat18$theta_y)
R> flat_theta50_64 <- sum(pe_flat50$theta_y)
R> marg_flat18 <- tapply(pe_flat18$theta_y,
     list(pe_flat18$gender, pe_flat18$edu_attain), sum)
R> marg_flat50 <- tapply(pe_flat50$theta_y,
     list(pe_flat50$gender, pe_flat50$edu_attain), sum)
R> round(marg_flat18 / flat_theta18_34, 4)
R> round(marg_flat50 / flat_theta50_64, 4)
```

The marginalized estimates of $\hat{\theta}$ with a non-informative prior show that education levels have improved over time as the younger cohort has higher estimated parameters for 'some college' or higher educational attainment. It is also clear that there is greater gender equality in educational outcomes for the younger cohort. But the analyst would not draw these conclusions using a strong flat prior. As expected, a strong flat prior exerts substantial smoothing effects on the estimates of $\hat{\theta}$. While there is a large range in parameter estimates for the non-informative prior, all estimates for the flat prior have been smoothed to near 0.07. This comparison shows that, as with most Bayesian analyses, the results of imputation of multinomial data can be quite sensitive to the choice of prior.

Ages 18-34	lt_hs	some_hs	hs_grad	some_col	assoc_dec	ba_deg	grad_deg
Female	0.0507	0.0421	0.1523	0.1251	0.0000	0.0912	0.0205
Male	0.0423	0.0928	0.0922	0.1141	0.0187	0.1197	0.0383
Ages 50-64							
Female	0.1260	0.0866	0.1304	0.1051	0.0405	0.0000	0.0318
Male	0.1704	0.0980	0.0357	0.0471	0.0575	0.0710	0.0000

Table 1: Estimates of the marginal distribution of $\hat{\theta}$ by gender and educational attainment using a non-informative prior. Estimates are for individuals aged 18-34 and 50-64.

Ages 18-34	lt_hs	some_hs	hs_grad	some_col	assoc_dec	ba_deg	grad_deg
Female	0.0684	0.0671	0.0831	0.0791	0.0611	0.0744	0.0641
Male	0.0671	0.0744	0.0744	0.0776	0.0641	0.0784	0.0668
Ages 50-64							
Female	0.0763	0.0729	0.0764	0.0744	0.0687	0.0653	0.0680
Male	0.0801	0.0738	0.0652	0.0694	0.0702	0.0713	0.0652

Table 2: Estimates of the marginal distribution of $\hat{\theta}$ by gender and educational attainment using a strong flat prior. Estimates are for individuals aged 18-34 and 50-64.

4. Comparing imputeMulti

In this section, we compare **imputeMulti** with other popular imputation methods. Specifically, we examine bootstrap EM via **Amelia** Honaker *et al.* (2011), polytomous logistic regression via **mice** van Buuren and Groothuis-Oudshoorn (2011), and predictive mean matching via **Hmisc** Harrell Jr and others. (2016). The goal of the comparison is not to provide an *exhaustive* comparison of imputation methods; but to compare commonly used methods within each package. The `tract2221` dataset is again used for illustration, in this case looking at imputation of five variables: `age`, `gender`, `marital_status`, `edu_attain`, and `emp_status`.

The first thing to note is that **Amelia** does not allow for categorical variables with greater than ten levels, while `tract2221$age` has sixteen levels.

```
R> library("Amelia")
R> amelia_EM <- amelia(tract2221[,1:5], m = 1, noms = 1:5)

Warning message:
In amcheck(x = x, m = m, idvars = numopts$idvars, priors = priors,  :

The number of categories in one of the variables marked nominal has greater
than 10 categories.
Check nominal specification.
```

This is one serious limitation on multivariate multinomial imputation when using a package designed for a multivariate normal distribution. Researchers using **Amelia** must immediately decide the best way to limit some of the richness of their data. None of the other three packages have this restriction. To keep the remainder of the comparison as equivalent as possible, only four variables are used: `gender`, `marital_status`, `edu_attain`, and `emp_status`.

4.1. Comparing algorithm speed

The **microbenchmark** package Mersmann (2015) is used for speed comparisons. Tests were performed on a Intel Xeon CPU E5-2960 v3 2.60GHz server running Windows Server 2012 R2 Standard and were not run in parallel. To purely test algorithm speed, multiple imputations are not specified for any packages that allow them.

```
R> library("Amelia")
R> library("Hmisc")
```

```
R> library("imputeMulti")
R> library("mice")
R> library("microbenchmark")
R> test_d <- tract2221[,2:5]
R> microbenchmark(
  EM = multinomial_impute(test_d, "EM", conj_prior = "non.informative"),
  DA = multinomial_impute(test_d, "DA", conj_prior = "non.informative"),
  amelia = amelia(test_d, m = 1, noms = 1:4),
  hmisc = aregImpute(~ gender + marital_status + edu_attain + emp_status,
                  data = test_d, n.impute = 1, type = "pmm"),
  mice = mice(test_d, m = 1, method = "polyreg"), times = 10L)

Unit: milliseconds
expr       min        lq      mean    median        uq       max    neval
EM      2271.97  2279.53  2336.20  2282.18  2418.99  2532.10    10
DA      3664.21  3843.10  4328.98  4242.36  4948.12  4979.40    10
amelia  144.29   149.73    175.99  151.40    153.20   402.19    10
hmisc   682.13   688.32    822.01  819.47    945.75   997.42    10
mice    3899.12 3913.96   4091.42 3956.81   4161.81  4995.85    10
```

All algorithms finish in a matter of milliseconds or seconds. **Amelia** is clearly the fastest algorithm, beating **imputeMulti** by more than an order of magnitude in this test. As expected, EM, which stops upon convergence, is noticeably faster than DA, which must run all `burnin` iterations before calculating posterior means.

4.2. Comparing outputs: Parameter estimates

Differences in the outputs of the various methods are examined next, focusing on differences in output rather than the the statistical properties of each package. For a discussion of using the multivariate normal to approximate multinomial data, see (Schafer 1997, Chapter 6). For the purposes of this article, a comparison of a single commonly used method within each package is used rather than an exhaustive comparison. As above, multiple imputations are not specified for any packages that allow them. Functions used for these comparisons as well as post-processing the **Hmisc** and **mice** outputs (Section 4.3) are provided in the supplementary material to this article.

```
R> source("./00_JSS_suplemental_functions.R")
R> set.seed(1987543L)
R> test_dat1 <- tract2221[complete.cases(tract2221[,2:5]), 2:5]
R> test_dat2 <- createNAs(test_dat1, pctNA = 0.15)
R> rownames(test_dat2) <- 1:nrow(test_dat2)

R> IM_EM <- multinomial_impute(test_dat2, "EM",
     conj_prior = "non.informative", verbose = TRUE)
R> amelia_EM <- amelia(test_dat2, m = 1, noms = c("gender",
     "marital_status", "edu_attain", "emp_status"))
R> hmisc_pmm <- aregImpute(~ gender + marital_status + edu_attain +
```

```
      emp_status, data = test_dat2, n.impute = 1, type = "pmm")
R> mice_ch_eq <- mice(test_dat2, m = 1, method = "polyreg")
```

Amelia, **mice**, and **Hmisc** all use S3 classes for outputs. In both **mice** and **Hmisc**, the parameter estimates are not directly available. The parameter estimates from the `amelia` function are found in `amelia_EM$mu` and `amelia_EM$theta`. This output structure fits the expectation that the data comes from a multivariate normal (eg. $Y \sim N(\mu, \Sigma)$). The output of `amelia_EM$mu` is shown below.

```
noms.gender.2         -0.034183160
noms.marital_status.2 -0.001370118
noms.marital_status.3  0.001611829
noms.marital_status.4 -0.005067206
noms.marital_status.5  0.040268816
noms.edu_attain.2      0.014664753
noms.edu_attain.3      0.011949303
noms.edu_attain.4      0.020386341
noms.edu_attain.5     -0.024465460
noms.edu_attain.6      0.015697896
noms.edu_attain.7      0.003025067
noms.emp_status.2     -0.022108630
noms.emp_status.3      0.012268149
```

As is clear from the above output, when working with multivariate multinomial data using the **Amelia** package, researchers are required to post-process parameter outputs to fit their needs. It is not immediately obvious how to get specific multinomial parameter estimates or how to marginalize these estimates, for example obtaining the marginal distribution of $\hat{\theta}$ by gender and educational attainment as in Section 3.4. An additional advantage to using **imputeMulti** is therefore improved ease of use for researchers interested in multinomial parameter estimates.

4.3. Comparing outputs: Observation level imputations

Observation level imputations are provided by **Amelia**, **mice**, and **Hmisc**. In **Amelia** and in **imputeMulti**, imputed observations are returned in the original dataset and can be compared directly. For both **mice** and **Hmisc** only the missing observations and their row numbers are returned. The imputations thus require post-processing to insert the imputed values into the original dataset containing missing values.

To test accuracy of observation level imputations, two tests are run. The first test does not allow for multiple imputations, while the second does. To setup the tests, the `complete.cases` from `tract2221[,2:5]` are extracted and missing values are randomly inserted. Observation level imputation is then conducted for these datasets and the imputations were compared to the original, fully observed, dataset. To test sensitivity to the amount of missingness in the data, tests are run using 15%, 30%, 45%, and 60% missing values.

The first test compares accuracy of **imputeMulti** to a single imputation in the other methods. The second test allows ten multiple imputations and compares the accuracy of a single **imputeMulti** imputation to the maximum accuracy across the ten multiple imputations using

Percent Missing	IM-EM	IM-DA	Amelia	Hmisc	mice
15%	0.7642	0.7615	0.6761	0.7077	0.7133
30%	0.5700	0.5639	0.4404	0.4833	0.4882
45%	0.4135	0.4077	0.2679	0.3166	0.3188
60%	0.2827	0.2794	0.1460	0.1908	0.1993

Table 3: Mean accuracy of observation level imputation for **imputeMulti**, **Amelia**, **Hmisc**, and **mice** using a single imputation.

Percent Missing	IM-EM	IM-DA	Amelia	Hmisc	mice
15%	0.7651	0.7615	0.6857	0.7175	0.7166
30%	0.5687	0.5645	0.4493	0.4968	0.4966
45%	0.4097	0.4070	0.2816	0.3233	0.3298
60%	0.2853	0.2857	0.1653	0.1994	0.2052

Table 4: Mean accuracy of observation level imputation for **imputeMulti**, **Amelia**, **Hmisc**, and **mice**. A single imputation from **imputeMulti** is compared to the maximum accuracy from ten multiple imputations of the comparison methods.

the other methods. Ten such tests are run for each level of inserted missingness and the mean results are shown in Table 3 and Table 4.

Unsurprisingly, imputation accuracy degrades for each method as missingness increases. But, in each test, **imputeMulti** has the highest accuracy for completely matching true observations; and, **imputeMulti** maintains this advantage even against multiple imputations. The closest comparison method was **mice**.

5. Conclusion

Deciding how to deal with missing values is a frequent concern for applied researchers. Although many methods exist for missing value imputation, the vast majority rely on assumptions of multivariate normality. It is often desirable to use a model specifically designed for the distribution from which a researcher's data is generated. **imputeMulti** provides an easily accessible model for imputing multivariate multinomial data. Further, **imputeMulti** integrates easily into common analyses and handles large datasets with high performance by providing functions for calculating sufficient statistics in C++. **imputeMulti** also allows parallel processing in the case of extremely large datasets.

This article provides a hands-on introduction to the **imputeMulti** package as well as comparison to existing methods in R for multivariate multinomial imputation. As shown, the use of a package specifically designed for multivariate multinomial imputation, such as **imputeMulti**, provides exact solutions for datasets with small numbers of variables; and, for datasets with large numbers of variables, **imputeMulti** may be used by splitting the variables into related groups. In addition, **imputeMulti** produces higher observation level imputation accuracy as well as providing easier access to parameter estimates.

Future development of **imputeMulti** is expected to focus on extending performance by migrating more of the code to C++, improving the search algorithms used in calculating the multinomial distribution sufficient statistics by implementing balanced-trees, and exploring

further parallelization of parameter estimates. As with any parallelization effort, the tradeoff between parallel overhead and increased utilization of multi-core computing resources must be intelligently balanced. To date, timing tests on Windows machines, which do not allow forking, have indicated that the parallel overhead is only justified for datasets with several hundred thousand or more observations.

References

Darnieder WF (2011). *Bayesian Methods for Data-Dependent Priors*. Ph.D. thesis, The Ohio State University.

Dempster AP, Laird NM, Rubin DB (1977). "Maximum likelihood from incomplete data via the EM algorithm." *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38.

Eddelbuettel D, Francois R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**, 1–18. doi:10.18637/jss.v040.i08.

Ferguson TS (1973). "A Bayesian analysis of some nonparametric problems." *The annals of statistics*, pp. 209–230.

Fuchs C (1982). "Maximum likelihood estimation and model selection in contingency tables with missing data." *Journal of the American Statistical Association*, pp. 270–278.

Harrell Jr FE, *et al.* (2016). ***Hmisc**: Harrell Miscellaneous*. R package version 3.17-4, URL https://CRAN.R-project.org/package=Hmisc.

Honaker J, King G, Blackwell M (2011). "**Amelia II**: A Program for Missing Data." *Journal of Statistical Software*, **45**(7), 1–47. URL http://www.jstatsoft.org/v45/i07/.

Kane MJ, Emerson J, Weston S (2013). "Scalable Strategies for Computing with Massive Data." *Journal of Statistical Software*, **55**(14), 1–19. URL http://www.jstatsoft.org/v55/i14/.

Mersmann O (2015). ***microbenchmark**: Accurate Timing Functions*. R package version 1.4-2.1, URL https://CRAN.R-project.org/package=microbenchmark.

Orcutt G, *et al.* (1961). *Microanalysis of Socioeconomic Systems: A Simulation Study*. Harper.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rubin D (1987). *Multiple Imputation for Nonresponse in surveys*. John Wiley & Sons.

Schafer JL (1997). *Analysis of incomplete multivariate data*. CRC Press.

Tanner MA, Wong WH (1987). "The calculation of posterior distributions by data augmentation." *Journal of the American statistical Association*, pp. 528–540.

van Buuren S, Groothuis-Oudshoorn K (2011). "**mice**: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software*, **45**(3), 1–67. URL http://www.jstatsoft.org/v45/i03/.

Warnes GR, Bolker B, Lumley T (2015). ***gtools**: Various R Programming Tools*. R package version 3.5.0, URL https://CRAN.R-project.org/package=gtools.

Wickham H, Chang W (2016). ***devtools**: Tools to Make Developing R Packages Easier*. R package version 1.10.0, URL https://CRAN.R-project.org/package=devtools.

Affiliation:

Alex Whitworth
Seattle, WA 98103
E-mail: whitworth.alex@gmail.com