

Multivariate amputation using `ampute`

Rianne Schouten [aut, cre], Peter Lugtig [ctb], Gerko Vink [aut]

Saturday, October 8th, 2016

When new multiple imputation techniques are tested, simulated data sets have to be made incomplete. Often, a univariate amputation procedure is followed, generating missing values one variable at a time. This procedure is repeated for every variable that should have missing values. Van Buuren (2012, pp. 63, 64) explains in detail how this can be done. However, there are some drawbacks of this approach. First, a univariate amputation procedure makes it difficult to relate the missingness on one variable to the missingness on any other variable. Second, both simulated and real data have a multivariate structure. Applying a univariate amputation procedure to multivariate data would not do justice to the complicated nature of data sets. For these reasons, function `ampute` is developed to perform multivariate amputation according to the user's specific desires.¹

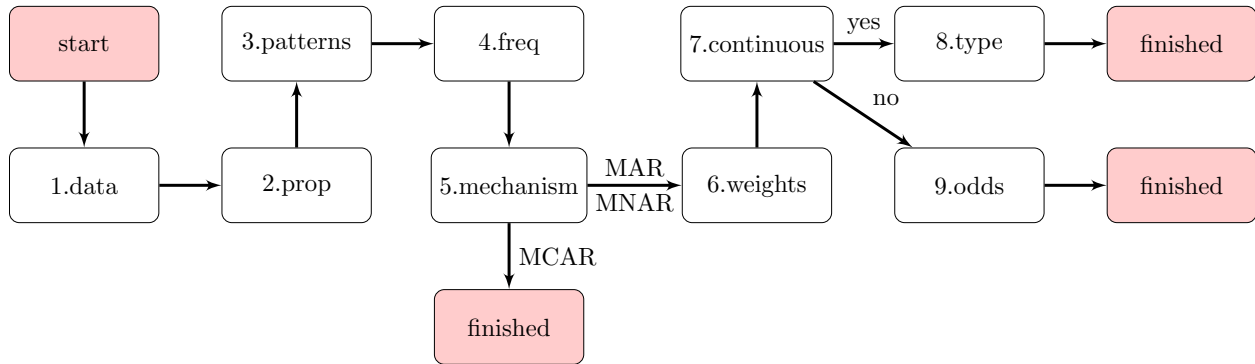


Figure 1: Step-by-step flowchart of R-function `ampute`

Function `ampute()` enables the generation of missing values based on different underlying missingness mechanisms. To this end, `ampute()` has multiple arguments and the most important ones are displayed in Figure 1. Figure 1 also gives an overview of the order in which decisions need to be taken if you would like to generate a missingness mechanism that cannot easily be traced back. This is especially useful for the evaluation and comparison of missing data methods. Nevertheless, the default settings create genuine multivariate missingness as well.

The underlying method of `ampute()` is based on Brand's (1999) multivariate amputation procedure. An adaptation to this method is made to make continuous amputation possible. The possibility to create MNAR missingness is an extra feature as well.

In this vignette, Figure 1 serves as a guideline to explain how `ampute`'s arguments can be used to generate multivariate missingness. We will discuss the possibilities by examining each step of Figure 1. Concurrently, we will explain the underlying amputation method since this might give the user the opportunity to not only generate missing values but also use the method's features to generate complicated missingness mechanisms. Our goal is to make the use of `ampute()` easy and straightforward.

Download latest version

To work with the latest version of `ampute()`, download the following documents from

<https://github.com/RianneSchouten/mice/tree/ampute/R>:

¹Since this pdf-document contains multiple layers, printing is possible after reducing the file (f.e. through www.smallpdf.com).

- ampute.R
- ampute.mcar.R
- ampute.continuous.R
- ampute.discrete.R
- ampute.default.R
- is.R
- print.R
- summary.R
- mads.R

To apply the plot functions, load:

```
require(lattice)
```

and download:

- xyplot.mads.R
- bwplot.mads.R

1. Data and quick amputation

First, we need a complete data set. In simulation settings, multivariate data can be generated by using `mvnrm()` from the package `MASS`. Be aware that the covariance matrix should be semi definite.

```
set.seed(2016)
testdata <- mvnrm(n = 10000, mu = c(10, 5, 0),
                  Sigma = matrix(data = c(1.0, 0.2, 0.2,
                                          0.2, 1.0, 0.2,
                                          0.2, 0.2, 1.0), nrow = 3, byrow = T))
testdata <- as.data.frame(testdata)
summary(testdata)
```

##	V1	V2	V3
## Min.	: 6.346	Min. :0.9789	Min. : -3.799124
## 1st Qu.:	9.317	1st Qu.:4.3199	1st Qu.: -0.690542
## Median :	10.003	Median :4.9963	Median : -0.009941
## Mean :	9.998	Mean :4.9973	Mean : 0.000759
## 3rd Qu.:	10.677	3rd Qu.:5.6759	3rd Qu.: 0.673069
## Max.	:13.770	Max. :8.6579	Max. : 3.810729

The function `ampute()` immediately works when the data are entered into the function. Storing the result allows you to work with the amputed data.

```
result <- ampute(testdata)
result

## Multivariate Amputed Data Set
## Call: ampute(data = testdata)
## Class: mads
## Proportion of Missingness: 0.5
## Frequency of Patterns: 0.3333333 0.3333333 0.3333333
## Pattern Matrix:
##   V1 V2 V3
## 1  0  1  1
## 2  1  0  1
## 3  1  1  0
## Mechanism:[1] "MAR"
```

```
## Weight Matrix:
##   V1 V2 V3
## 1  0  1  1
## 2  1  0  1
## 3  1  1  0
## Type Vector:
## [1] "RIGHT" "RIGHT" "RIGHT"
## Odds Matrix:
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    1    2    3    4
## [3,]    1    2    3    4
## Head of Amputed Data Set
##      V1      V2      V3
## 1      NA 6.018999 0.3681981
## 2  9.668991 4.391821 -1.1127595
## 3 10.273415 4.662521  0.1796964
## 4 10.124800 4.055544      NA
## 5      NA 7.171578  1.7141378
## 6 10.803311 5.038649      NA
```

Class

The return object is of class `mads` (Multivariate Amputed Data Set) and contains all the useful information about the amputation. In total, the object contains the following:

```
names(result)
```

```
## [1] "call"      "prop"      "patterns"  "freq"      "mech"      "weights"
## [7] "cont"      "type"      "odds"      "amp"       "cand"      "scores"
## [13] "data"
```

Inspect amputed data

The amputed data is stored under `amp`. To see whether the amputation has gone according plan, a quick investigation can be done by using function `md.pattern()`.

```
md.pattern(result$amp)
```

```
##      V3  V2  V1
## 5020   1   1   1   0
## 1675   1   1   0   1
## 1665   1   0   1   1
## 1640   0   1   1   1
##      1640 1665 1675 4980
```

The rows of the table show the different missing data patterns with the number of cases accordingly. The first row always refers to the complete cases. The last column contains the number of variables with missings in that specific pattern. Consequently, each column total describes the number of cells with missing values for that variable. A more thorough explanation of `md.pattern()` can be found in its help file (`?md.pattern`). Note that because `md.pattern()` sorts the columns in increasing amounts of missing information, the order of the variables is different from the order in the data.

2. Proportion of missingness

The proportion of missingness is specified under:

```
result$prop
```

```
## [1] 0.5
```

In the default setting, this means that 50% of the cases will have missing values. It is easy to change this proportion by using the argument `prop`. One might also want to specify the percentage of missing cells. For this, the argument `bycases` should be `FALSE`.

```
result <- ampute(testdata, prop = 0.2, bycases = FALSE)
md.pattern(result$amp)
```

```
##      V3  V1  V2
## 4036   1   1   1   0
## 1989   1   0   1   1
## 2004   1   1   0   1
## 1971   0   1   1   1
##      1971 1989 2004 5964
```

An inspection of the result shows that the proportion of missing cells is approximately 20%, as requested (the data set contains $10000 * 3 = 30000$ cells, in total, 5964 cells are made missing). `ampute()` automatically calculates the proportion of missing cases that belongs to this setting.

```
result$prop
```

```
## [1] 0.6
```

3. Patterns

The basic idea of `ampute()` is the generation of missingness patterns. Each pattern is a combination of missingness on specific variables while other variables remain complete. For example, someone could have forgotten the last page of a questionnaire, resulting in missingness on a specific set of questions. Another missingness pattern could occur when someone is not willing to answer private questions. Or when a participant misses a wave in a longitudinal study. Consequently, each pattern is a specific combination of missing and complete variables.

The default missingness patterns can be obtained by:

```
mypatterns <- result$patterns
mypatterns
```

```
##   V1 V2 V3
## 1  0  1  1
## 2  1  0  1
## 3  1  1  0
```

In the `patterns` matrix, each row refers to a missing data pattern and each column to a variable. 0 is used for variables that should have missing values in a particular pattern. 1 is used otherwise. Here, three missing data patterns are specified with missing values on 1 variable only. Note that as a result of this, none of the cases will have missingness on more than one variable. A case either has missingness on V1, V2 or V3 or remains complete.

Subsequently, the default `patterns` matrix can be changed according your desires. For example, the missingness patterns might be changed into:

```

mypatterns[2, 1] <- 0
mypatterns <- rbind(mypatterns, c(0, 1, 0))
mypatterns

```

```

##   V1 V2 V3
## 1  0  1  1
## 2  0  0  1
## 3  1  1  0
## 4  0  1  0

```

By doing this, a missingness pattern is created where cases will have missingness on V1 and V2 but not on V3 (pattern 2). Also, I have added a fourth missing data pattern to create a combination of missingness on V1 and V3.

Now, I can ampute the data again, with the desired `patterns` matrix as its third argument. Note that I have changed the desired proportion of missingness to `bycases = TRUE` (default) and 30%.

```

result <- ampute(testdata, prop = 0.3, patterns = mypatterns)
md.pattern(result$amp)

```

```

##      V2   V3   V1
## 6996  1    1    1    0
## 721   1    1    0    1
## 729   1    0    1    1
## 806   0    1    0    2
## 748   1    0    0    2
##      806 1477 2275 4558

```

4. Relative frequency

The function `ampute()` works by assigning cases to the different missing data patterns. Each case is assigned to one pattern only, and all cases are divided among the patterns. In other words, every case is *candidate* for a certain missing data pattern. This does not automatically mean that each case will obtain missing values. That decision will be made later on.

The argument `freq` specifies with which frequency the cases should be divided over the patterns. As a default, equal frequencies are used for the patterns.

```

result$freq

```

```

## [1] 0.25 0.25 0.25 0.25

```

The specifications below are an example of a situation where one would like to impose missing data pattern 1 with a higher frequency than the other missing data patterns. When changing the `freq` argument, one should keep in mind that the sum of the relative frequencies should always be 1 (in order to divide all the cases over the patterns).

```

result <- ampute(testdata, prop = 0.3, patterns = mypatterns,
                  freq = c(0.7, 0.1, 0.1, 0.1))
md.pattern(result$amp)

```

```

##      V2   V3   V1
## 6958  1    1    1    0
## 2131  1    1    0    1
## 318   1    0    1    1
## 288   0    1    0    2
## 305   1    0    0    2

```

```
##      288 623 2724 3635
```

An inspection of the missingness patterns using `md.pattern()` shows that indeed pattern 1 has received 7 times as many candidates as pattern 2 to 4. In total, approximately 3000 cases have missing values (which is what was specified under `prop`).

5. Mechanism

At this point, the total proportion of missingness is defined, the missing data patterns are specified as well as the relative frequency with which they should occur. All cases are candidate for a missing data pattern.

Whether a case will be made missing eventually depends on the missingness mechanism. If each case should have an equal probability of having missing values, the argument `mech` should be changed to "MCAR" (Missing Completely At Random). Then, step 6 to 9 do not need to be specified.

```
result <- ampute(testdata, prop = 0.3, patterns = mypatterns,
                 freq = c(0.7, 0.1, 0.1, 0.1), mech = "MCAR")
result$mech
```

```
## [1] "MCAR"
```

Two other options are "MAR" (Missing At Random), where the probability to be missing depends on the values of the variables that will not be made missing (the 'observed' variables) or "MNAR" (Missing Not At Random), where these probabilities depend on the values of the variables that will be made missing. For a more thorough explanation of these terms, Van Buuren (2012, pp. 6, 7, 31, 32) is useful.

6. Weights

In case of MAR (default) or MNAR missingness, a weighted sum score will be calculated for each case. This is an important part in the process because the probability that a case will be made missing depends on this score. The way in which the sum scores are used, will be explained in step 8 and step 9.

The weighted sum scores are built from the variable values and certain pre-specified weights. For each case, the value on a certain variable is multiplied with a weight. This is done for all variables and the resulting values are summed: a weighted sum score. The formula that describes the calculation is:

$$s_{ik} = \sum_{j=1}^J w_{jk} * c_{ij} \quad (1)$$

where s_{ik} is the weighted sum score of a case i in a certain pattern k , w_{jk} is the pre-specified weight of a certain variable j in a certain pattern k and c_{ij} is the value of case i on variable j . In the example, $j \in \{1, 2, 3\}$ and $k \in \{1, 2, 3, 4\}$ because there are three variables and four missing data patterns.

The `weights` matrix stores the w_{jk} and is of size `#patterns` by `#variables`. The default `weights` matrix for MAR missingness is as follows.

```
result <- ampute(testdata, prop = 0.3, patterns = mypatterns,
                 freq = c(0.7, 0.1, 0.1, 0.1))
myweights <- result$weights
myweights
```

```
##   V1 V2 V3
## 1  0  1  1
## 2  0  0  1
## 3  1  1  0
## 4  0  1  0
```

At first sight, this matrix might be complicated, but if you compare the matrix with the `patterns` matrix that we used, the contents make more sense.

```
mypatterns
```

```
##   V1 V2 V3
## 1  0  1  1
## 2  0  0  1
## 3  1  1  0
## 4  0  1  0
```

As you see, the `weights` and `patterns` matrices are exactly similar. The reason for this is that in case of MAR missingness, the variables that will be made missing should not be weighted (0 in the `weights` matrix). Those are the variables with 0 in the `patterns` matrix.

The other variables receive a weight of 1, as a default. Thus, the variables have equal weights.

In case of MNAR missingness, the default `weights` matrix is as follows:

```
result <- ampute(testdata, prop = 0.3, patterns = mypatterns,
                 freq = c(0.7, 0.1, 0.1, 0.1), mech = "MNAR")
result$weights
```

```
##   V1 V2 V3
## 1  1  0  0
## 2  1  1  0
## 3  0  0  1
## 4  1  0  1
```

If this matrix is compared with the `patterns` matrix that was used, it is easy to see that the values are reversed. Since a MNAR missingness mechanism means that the the missingness will depend on the variables that will be amputated, those variables are weighted in the `weight` matrix. Those are the variables with a value 0 in the `patterns` matrix. Again, the weighted variables receive equal weights of 1. The variables that will remain complete are not weighted and receive a 0.

Naturally, the idea of the `weights` matrix is to weight variables differently from each other. From now on, we will impose a MAR mechanism to the data. We therefore focus at the default `weights` matrix which we stored under `myweights`. For instance, we could desire to give the values on variable V2 a higher weight than the values on variable V3. Therefore, for pattern 1, we could change the `weights` matrix into something as:

```
myweights[1, ] <- c(0, 0.8, 0.4)
```

By choosing the values 0.8 and 0.4, variable V2 is weighted twice as heavy as variable V3. The weight values are relative values, meaning that choosing the values 8 and 4 would have the same effect on the amputation process. In order to clearly see the result of the `weights` setting, one can specify weights for just a few variables and with a relative big distance from each other.

For pattern 3, variable V1 will be weighted three times as heavy as variable V2.

```
myweights[3, ] <- c(3, 1, 0)
myweights
```

```
##   V1  V2  V3
## 1  0 0.8 0.4
## 2  0 0.0 1.0
## 3  3 1.0 0.0
## 4  0 1.0 0.0
```

We will now apply these settings and inspect the results in two ways: boxplots and scatterplots.

```
result <- ampute(testdata, prop = 0.3, patterns = mypatterns,
                 freq = c(0.7, 0.1, 0.1, 0.1), weights = myweights)
```

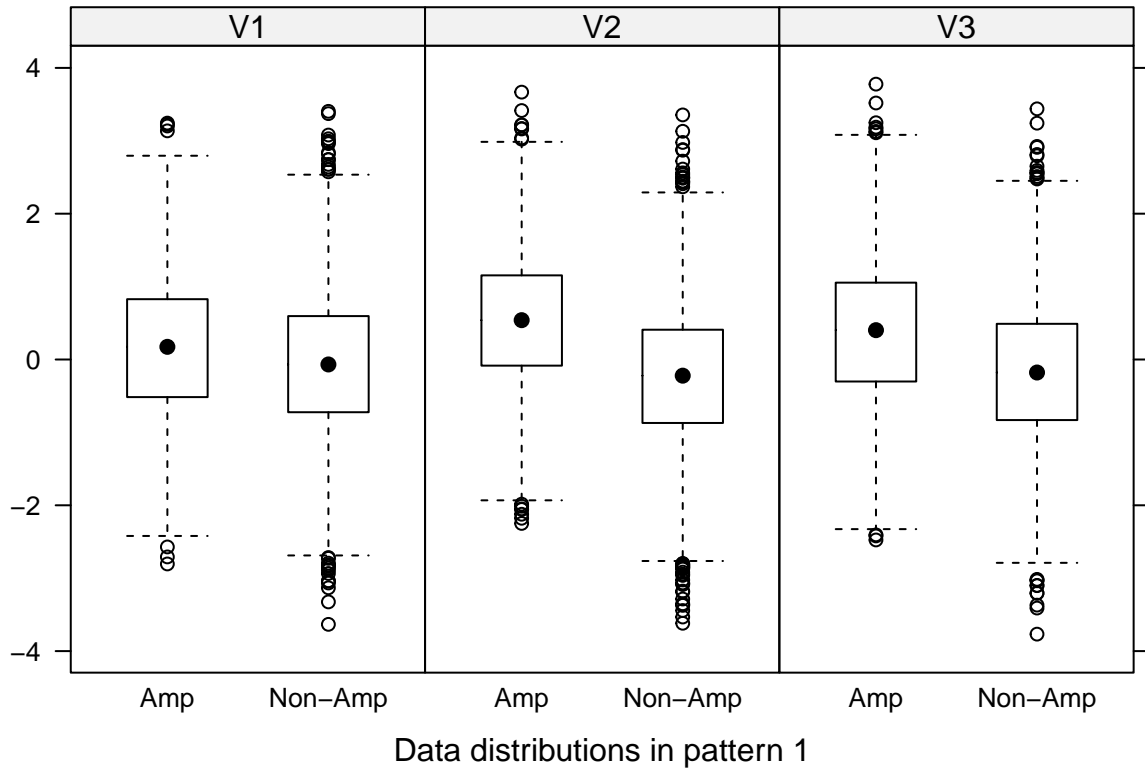
Boxplots

First, there is the function `bwplot()` that can be imposed on the `mads` object immediately. `bwplot()` plots the distributions of the amputed and non-amputed data for several variables. This is useful since these boxplots show the relation between the missingness and the variables values. In other words, by examining the boxplots one can see for which values of a certain variable the data will be amputed. Note that not necessarily the values of the variables themselves will be amputed. The boxplots merely show the relation between the amputations and the variables.

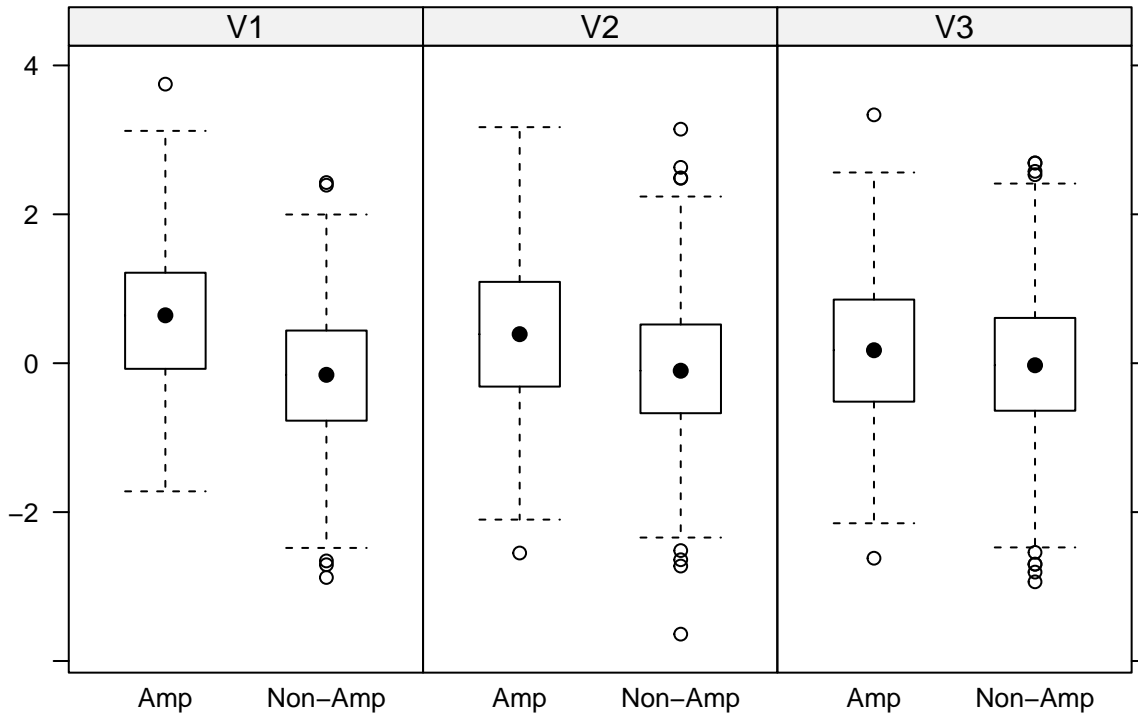
In the function `bwplot()`, the argument `which.pat` can be used to define the patterns you are interested in (default: all patterns). The argument `yvar` should contain the variables names (default: all variables). Besides, the function returns the mean, variance and n of the amputed and non-amputed data for each variable and each pattern requested. In the column `Amp`, a 1 refers to the amputed data and 0 to the non-amputed data. If the descriptives are not required, the argument `descriptives` can be set to `FALSE`.

```
bwplot(result, which.pat = c(1, 3), descriptives = TRUE)
```

```
## $Descriptives
## , , Variable = V1
##
##           Descriptives
## Pattern Amp      Mean      Var      N
##      1   1  0.14915 0.95221 2099
##      1   0 -0.06558 0.98711 4896
##      3   1  0.62349 0.89290  304
##      3   0 -0.16760 0.83478  688
##
## , , Variable = V2
##
##           Descriptives
## Pattern Amp      Mean      Var      N
##      1   1  0.52879 0.85752 2099
##      1   0 -0.22978 0.89401 4896
##      3   1  0.37603 0.97649  304
##      3   0 -0.09058 0.87815  688
##
## , , Variable = V3
##
##           Descriptives
## Pattern Amp      Mean      Var      N
##      1   1  0.38719 0.99257 2099
##      1   0 -0.17480 0.94550 4896
##      3   1  0.15376 0.94016  304
##      3   0  0.01534 0.96588  688
##
##
## $`Boxplot pattern 1`
```

```
##
## $`Boxplot pattern 3`
```



Data distributions in pattern 3

The medians and boundaries of the boxes show that in pattern 1, the amputated data is shifted to the right with respect to the non-amputated data. For variable V2, this effect is the largest, due to the weight value that was specified. For V1, there is a very small difference between the boxplots of the amputated and non-amputated data. This makes sense, because variable V1 was amputated in the first pattern and therefore set to 0 in the `weights` matrix. The small difference that is visible is due to the positive correlation between V1 on the one side and V2 and V3 on the other side. These correlations were created during the simulation of the data.

If desired, one could use the function `tsum.test()` from package `BSDA` to perform a t-test on the amputated and non-amputated data. The data returned in the descriptives table can be used for that. For example, to know whether the mean difference between the amputated and non-amputated data for variable V2 in pattern 1 is significant, one could run:

```
BSDA::tsum.test(mean.x = 0.52879, mean.y = -0.22978,
                 s.x = sqrt(0.85752), s.y = sqrt(0.89401),
                 n.x = 2099, n.y = 4896)
```

```
##
##  Welch Modified Two-Sample t-Test
##
## data: Summarized x and y
## t = 31.2, df = 4046.1, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.7109025 0.8062375
## sample estimates:
## mean of x mean of y
##  0.52879 -0.22978
```

As is visible, there is a significant difference between the amputated and non-amputated data of variable V2 in pattern 1.

For pattern 3, the difference between the distributions of the amputated and non-amputated data is largest for variable V1, as can be expected due to the weight values in pattern 3.

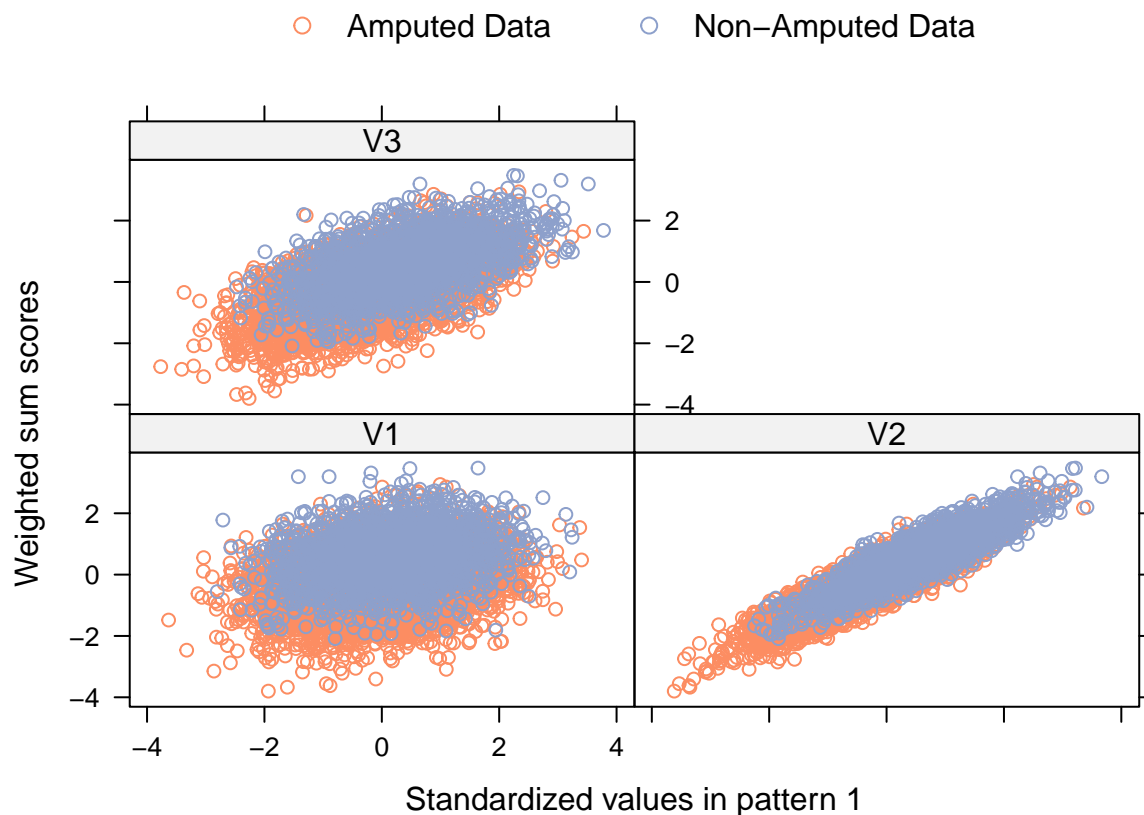
Scatterplots

Scatterplots might also help to investigate the effect of the specifications. We can directly impose the function `xyplot()` on the `mads` object. The function contains arguments comparable to `bwplot()`.

For example, we can investigate the weighted sum scores of pattern 1 as follows:

```
xyplot(result, which.pat = 1)
```

```
## $`Scatterplot Pattern 1`
```



The scatterplots show that there is a very small relation between V1 and the weighted sum scores. Furthermore, the relation between V2 and the weighted sum scores is very strong, meaning that a case's value on V2 is very important in the generation of the weighted sum score. Actually, this is what causes the differences between the amputed and non-amputed data in the boxplots above. For V3 and the weighted sum scores, the relation is a bit weaker than for V2 but more present than for V1.

7. Continuous

As a default, `ampute()` creates continuous missingness. This means that logit probability functions are used to define a candidate's probability of having missing values. The type of continuous amputation can be

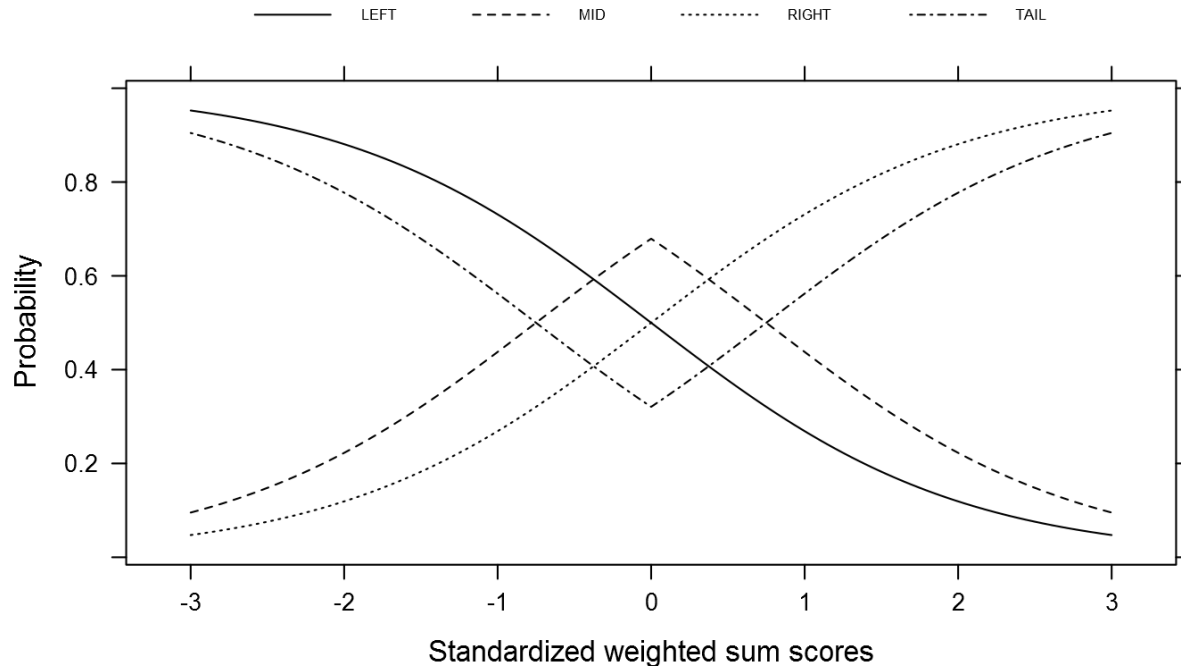


Figure 2: Adaptations of continuous logit functions

decided under `type`. A more thorough explanation of how the logit functions work will be given in part 8 of this vignette.

Instead of using continuous formulas to specify the missingness probabilities, these probabilities can also be defined by hand. Thereto, the argument `cont` should be set to `FALSE`. The `odds` argument can be used to define the probabilities (see part 9).

8. Type

The logit functions are more thoroughly explained by Van Buuren (2012, pp. 63, 64), but a quick overview will be given here. Four missingness functions are known: `RIGHT`, `MID`, `TAIL` and `LEFT` missingness. Figure 2 shows the course of the probabilities for standardized values.

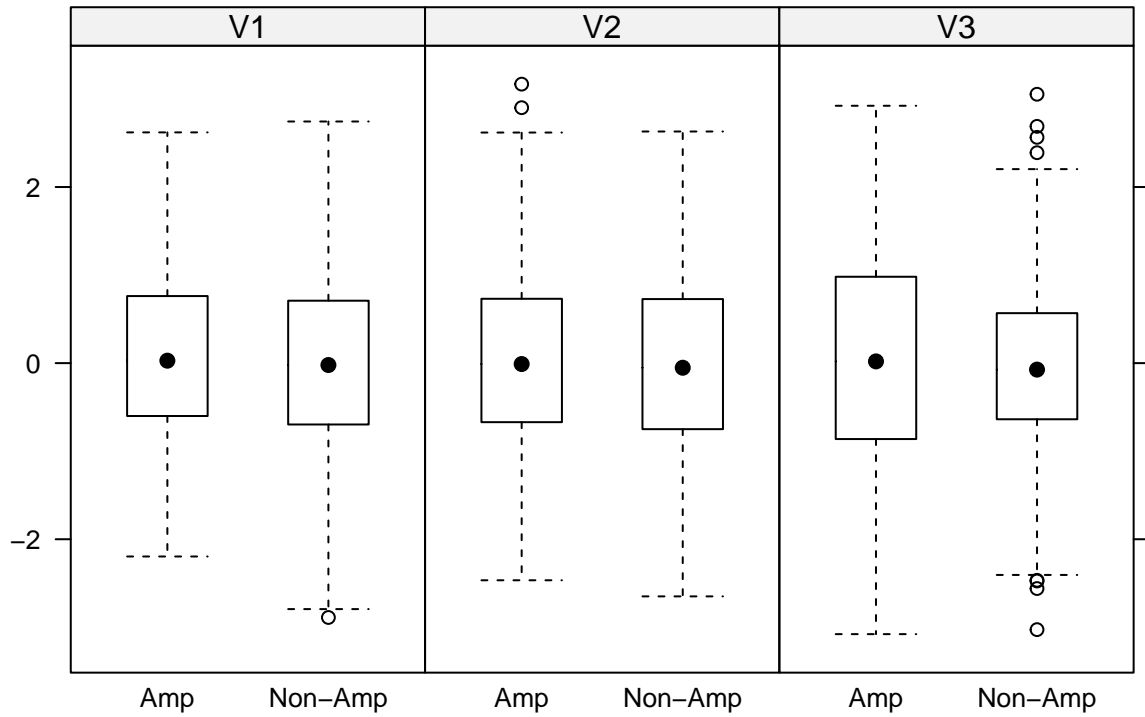
In `ampute()`, the logit functions will be applied to the weighted sum scores. Consequently, in the situation of `RIGHT` missingness, cases with high weighted sum scores will have a higher probability to have missing values, compared to cases with low weighted sum scores. For `MID` missingness, the higher probabilities are given to the cases with weighted sum scores around the average.

For each pattern, a different missingness type can be chosen. In our example, we have four patterns, so four type specifications are required. It is advised to inspect the result with `bwplot()` (below this is done for pattern 2), although the scatterplots give insight as well (as an example, a plot for pattern 4 is shown).

```
result <- ampute(testdata, prop = 0.3, patterns = mypatterns,
  freq = c(0.7, 0.1, 0.1, 0.1), weights = myweights,
  type = c("RIGHT", "TAIL", "MID", "LEFT"))
```

```
bwplot(result, which.pat = 2, descriptives = FALSE)
```

```
## $`Boxplot pattern 2`
```

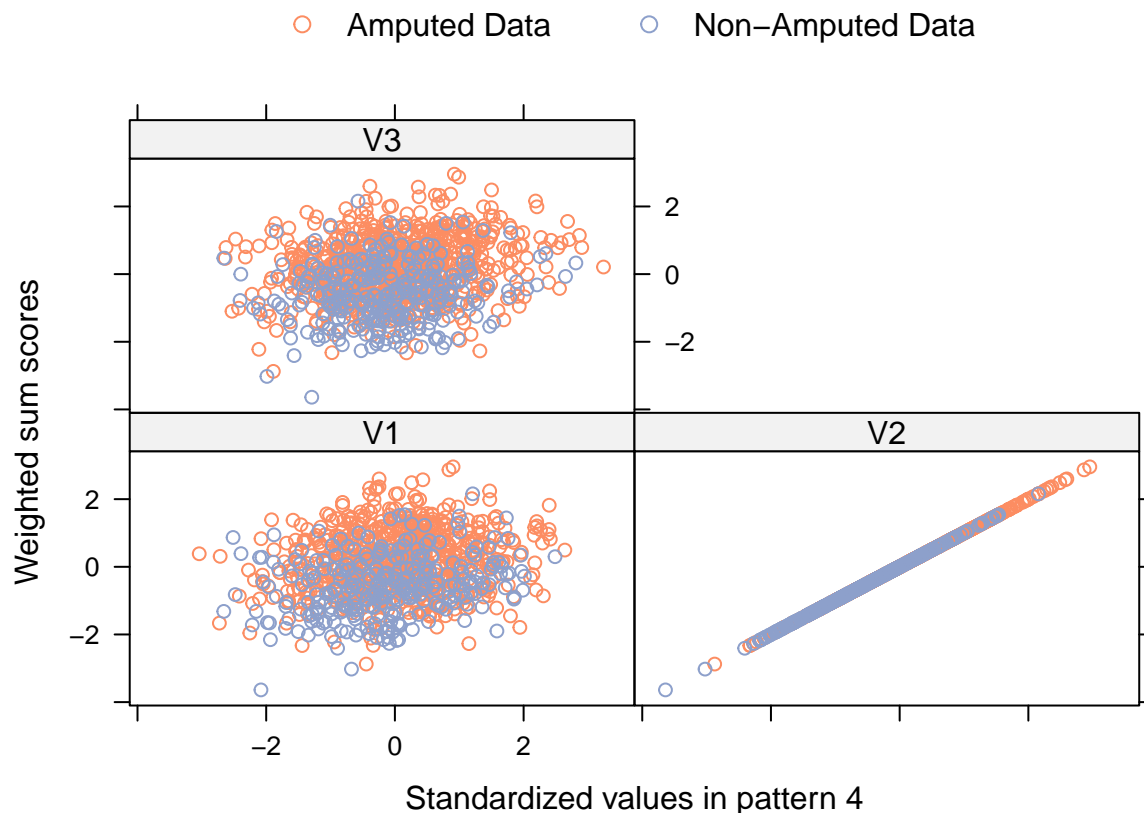


Data distributions in pattern 2

From the boxplots of pattern 2, it becomes visible that the interquartile range (IQR) is much larger for the amputated V3 values compared to the non-amputated data. This is due to the fact that in pattern 2 only V3 defines the missingness. Besides, we requested a TAIL missingness type, which means that all the cases with values at the tails of the distribution of the weighted sum scores (based on merely V3), will be made missing.

```
xyplot(result, which.pat = 4)
```

```
## $`Scatterplot Pattern 4`
```



First, notice that there are much fewer dots in these scatterplots compared to the scatterplots we saw earlier. This is due to the `freq` setting: we specified that only 10 percent of the cases with missing values should have missingness pattern 4. Second, the scatterplots show that all the amputed data are at the left hand side of the weighted sum scores due to the "LEFT" setting in the `type` argument. Third, these figures show that there is a perfect relation between variable V2 and the weighted sum scores. Clearly, pattern 4 depends on variable V2 only, which we can remember from the `weights` matrix we used.

```
result$weights
```

```
##   V1  V2  V3
## 1  0 0.8 0.4
## 2  0 0.0 1.0
## 3  3 1.0 0.0
## 4  0 1.0 0.0
```

9. Odds

If the missingness probabilities should not depend on a continuous probability function, you will have to define the probabilities to be missing yourself. This method was first described by Brand (1999) and consists of two steps. First, one has to decide in how many groups the weighted sum scores should be divided. Second, for each group, an odds value defines the relative probability of having missing values. In other words, each candidate case will be assigned to a group based on his weighted sum score.

Let us first have a look at the working of the odds values. The default `odds` matrix is as follows:

```
myodds <- result$odds
myodds
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    1    2    3    4
## [3,]    1    2    3    4
## [4,]    1    2    3    4
```

This matrix means that for each pattern, the candidates will be divided into four groups. This division occurs based on quantiles, in order to obtain equally sized groups. The values `c(1, 2, 3, 4)` mean that a case with a weighted sum score in the highest quantile will have a probability of having missing values that is four times higher than a candidate with a weighted sum score in the lowest quantile. In Figure 3 the different probabilities that belong to this setting are shown for 100 candidates of pattern 1.

As can be seen, there are indeed four groups in pattern 1. The groups have an approximately equal size, with each a certain probability to obtain missing values. The probability of group 4 is indeed four times as large as the probability of group 1.

The relation between the groups and the variable values is shown in Figure 4. Because the relationship between variable V2 and the weighted sum scores is high (due to the `weights` setting), the groups can be distinguished very well. Besides, for higher values of V2, the weighted sum scores are higher. These are the cases that are placed in group 4. Therefore, they are at the right hand side of the V2 scale. For variable V3, the relation between the values and the group allocation is small. This again is due to the `weights` setting. Still, because of the odds values, group 4 is much more to the right of the V3 scale than group 1, 2 and 3.

Let us now dig deeper into the contents of the `odds` matrix. The `#rows` of this matrix is equal to `#patterns`. The `#columns` can be defined by the user and depends on the desired amputation procedure.

The amount of values in each row defines the number of groups that will be created from the weighted sum scores in that specific pattern. This number can be different for the different patterns. The values themselves define the odds probabilities of having missing values. Note that the values are relative values. For instance,

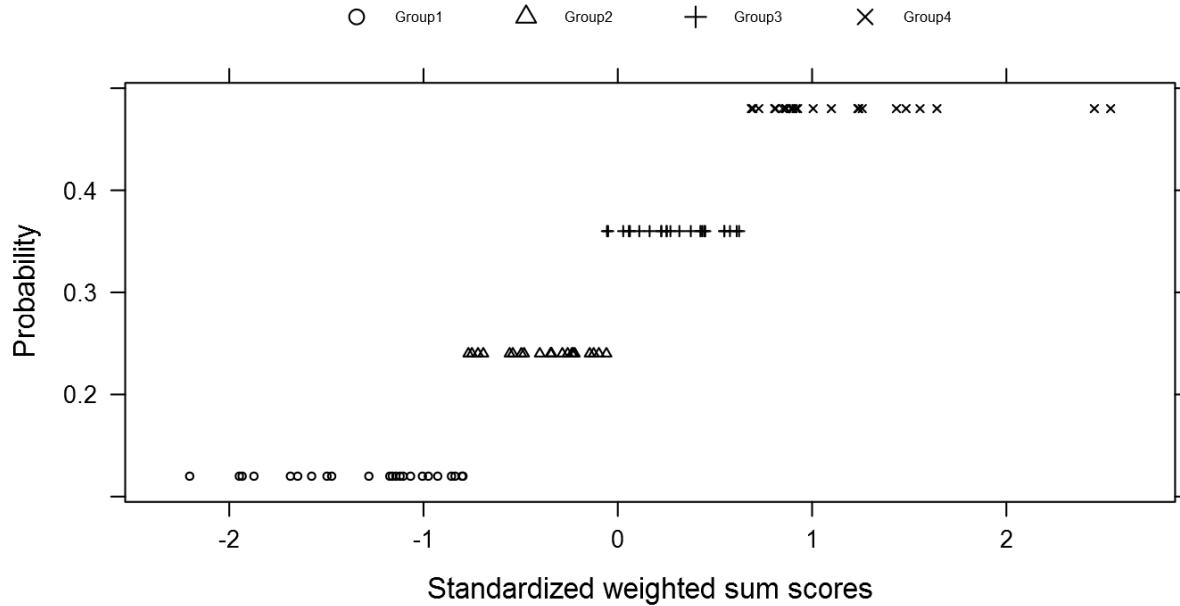


Figure 3: Probabilities to have missing values for the four groups in pattern 1 and the relation between the groups and the weighted sum scores.

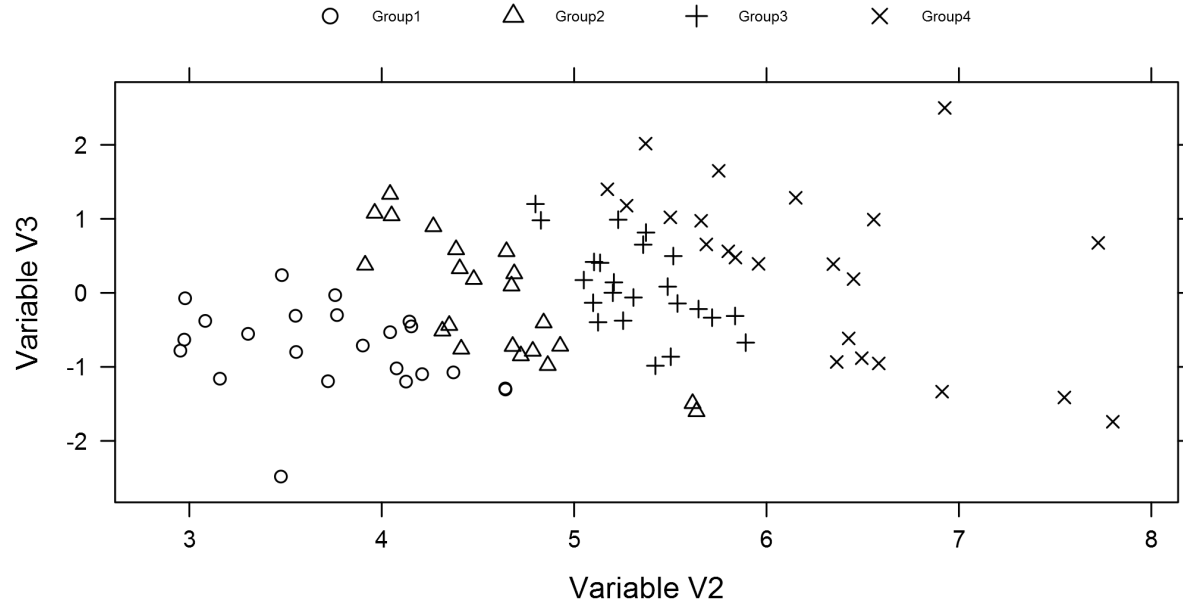


Figure 4: Division of odds groups over variables V_2 and V_3

a setting of $c(2, 6)$ is similar to $c(3, 9)$.

The cells in the `odds` matrix that are not used, should be filled with NAs. Let us define the matrix as follows.

```
myodds[3, ] <- c(1, 0, 0, 1)
myodds[4, ] <- c(1, 1, 2, 2)
myodds <- cbind(myodds, matrix(c(NA, NA, NA, 1, NA, NA, NA, 1), nrow = 4, byrow = F))
myodds
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  1    2    3    4    NA    NA
## [2,]  1    2    3    4    NA    NA
## [3,]  1    0    0    1    NA    NA
## [4,]  1    1    2    2    1    1
```

We keep the default setting of the first two patterns. Then, for pattern 3, the weighted sum scores will be divided into four groups. The odds values mean that candidates with low weighted sum scores will have a probability to have missing values that is equal to the probability of candidates with high weighted sum scores. However, candidates with weighted sum scores around average will not be made missing. Because pattern 3 depends on variable V_1 with a weight of 3 and on variable V_2 with a weight of 1, the effect will be most visible for variable V_1 .

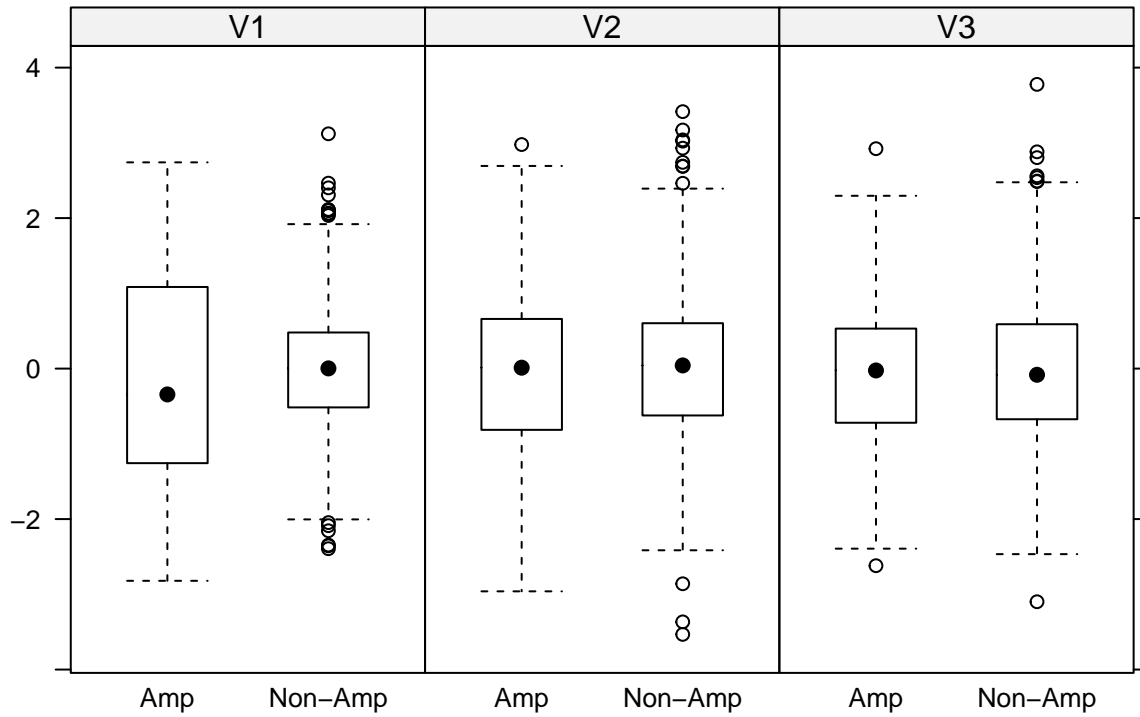
The weighted sum scores of the fourth pattern will be divided into six groups. All candidates will have a probability of having missing values, but this chance is larger for candidates with weighted sum scores around average.

```
result <- ampute(testdata, prop = 0.3, patterns = mypatterns,
  freq = c(0.7, 0.1, 0.1, 0.1), weights = myweights,
  cont = FALSE, odds = myodds)
```



```
bwplot(result, which.pat = c(3, 4), descriptives = FALSE)
```

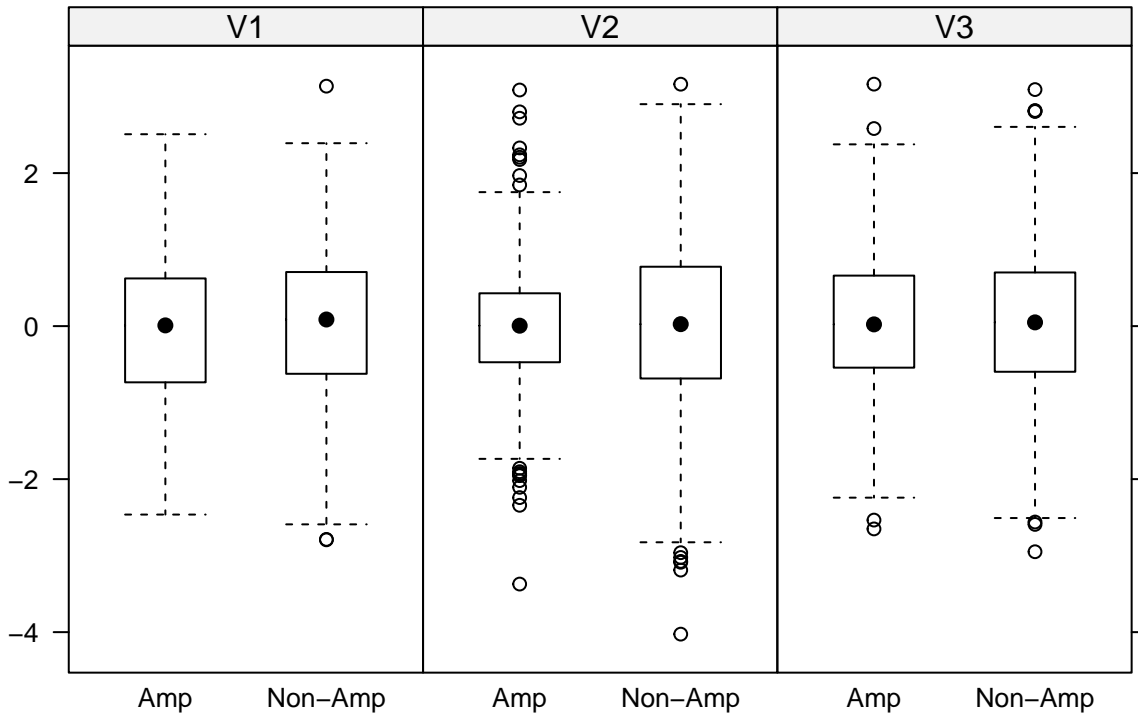
```
## $`Boxplot pattern 3`
```



Data distributions in pattern 3

```
##
```

```
## $`Boxplot pattern 4`
```



Data distributions in pattern 4

In the boxplots of pattern 3, it is visible that the IQR of the amputated data is larger than the one of the non-amputated data. This is especially the case for variable V1, a bit less for variable V2 but almost not present for variable V3.

In pattern 4, the effect of the specifications is only visible for variable V2, because the other variables will be made missing. In contrast to pattern 3, the amputation is performed in the center of the weighted sum scores, resulting to a MID-like missingness pattern.

10. Combining missingness mechanisms

Function `ampute()` has a few restrictions. For instance, all patterns should be generated based on either continuous (`cont == TRUE`) or discrete (`cont == FALSE`) probability distributions.

Furthermore, it is not yet possible to generate a MCAR mechanism together with a MAR or MNAR mechanism. On the other hand, the combination of MAR and MNAR is already possible.

A combination of MAR and MNAR missingness can be generated by changing the `weights` matrix. The following command, for instance, generates MAR missingness in pattern 1 and MNAR missingness in pattern 2. The total amputated dataset will contain both type of missingness mechanisms.

```
result <- ampute(testdata, prop = 0.3, patterns = c(0, 0, 1,
                                                    0, 1, 0),
                freq = c(0.7, 0.3), weights = c(0, 0, 1,
                                                  1, 0, 1))
```

Logically, it is possible to generate spurious MNAR mechanism by imposing weights on both observed and unobserved variables.

Unfortunately, the generation of both MCAR and MAR (or any form of spurious MAR) is not yet directly possible. By running the function twice, however, the desired combination of mechanisms can still be obtained.

Below is an example of how this can be done.

```
ampdata1 <- ampute(testdata, patterns = c(0, 1, 1), prop = 0.2, mech = "MAR")$amp
ampdata2 <- ampute(testdata, patterns = c(1, 1, 0), prop = 0.8, mech = "MCAR")$amp

indices <- sample(x = c(1, 2), size = nrow(testdata), replace = TRUE,
                  prob = c(1/2, 1/2))

ampdata <- matrix(NA, nrow = nrow(testdata), ncol = ncol(testdata))
ampdata[indices == 1, ] <- as.matrix(ampdata1[indices == 1, ])
ampdata[indices == 2, ] <- as.matrix(ampdata2[indices == 2, ])

md.pattern(ampdata)
```

```
##      [,1] [,2] [,3] [,4]
## 4938    1    1    1    0
## 1002    1    0    1    1
## 4060    1    1    0    1
##           0 1002 4060 5062
```

As can be seen with `md.pattern()`, 20% of cases is amputed with the first type of settings and 80 percent with the second type of settings. Here, for each mechanism, merely one missingness pattern is created to clearly see the difference between the two amputation rounds. Obviously, one could generate multiple patterns within one round.

Besides, these rounds are imposed on the data with equal probabilities (1/2 each). Consequently, 1/2 of 20% of the cases has a MAR missingness mechanism and 1/2 of 80% of the cases has a MCAR missingness mechanism. In total, this results in 50% of the cases having missing values.

11. Other specifications

Argument run

For large data sets or slow computers, you might not want to perform the amputation right away. When the argument `run` is set to `FALSE`, all results will be stored in the `mads` object except for the amputed data set. Subsequently, the default settings for the `patterns`, `weights` or `odds` argument can be changed easily and entered into a new run thereafter (with `run == TRUE`).

```
emptyresult <- ampute(testdata, run = FALSE)
emptyresult$amp
```

```
## data frame with 0 columns and 0 rows
```

Other mads contents

The return object from `ampute()` is of class `mads`. `mads` contains the amputed data set, the function specifications and some extra objects that might be useful.

The object `cand` is a vector that contains for every case the missing data pattern it was candidate for.

```
result$cand[1:30]
```

```
## [1] 1 1 2 1 1 2 1 2 1 1 1 1 2 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2
```

The object `scores` is a list with, for each pattern, the weighted sum scores of the candidates.

```
result$scores[[1]][1:10]
```

```
##           1           2           4           5           7           9
## 0.3642853 -1.1039606 0.2091449 1.6986723 1.4343838 -1.0476095
##           10          11          12          15
## -0.5556416 0.8364393 0.1252278 -0.7123034
```

Furthermore, the object `data` contains the original data.

```
head(result$data)
```

```
##           V1           V2           V3
## 1 10.487466 6.018999 0.3681981
## 2  9.668991 4.391821 -1.1127595
## 3 10.273415 4.662521 0.1796964
## 4 10.124800 4.055544 0.2117145
## 5 11.835097 7.171578 1.7141378
## 6 10.803311 5.038649 -0.2625144
```

Go ahead and ampute!

References

Brand, J.P.L. (1999). *Development, implementation and evaluation of multiple imputation strategies for the statistical analysis of incomplete data sets* (pp. 110-113). Dissertation. Rotterdam: Erasmus University.

Van Buuren, S. (2012). *Flexible imputation of missing data*. Boca Raton, FL.: Chapman & Hall/CRC Press.