missRows

Handling Missing Rows in Multi-Omics Data Integration

Valentin Voillet and Ignacio González

October 26, 2021

Abstract

In omics data integration studies, it is common, for a variety of reasons, for some individuals to not be present in all data tables. Missing row values are challenging to deal with because most statistical methods cannot be directly applied to incomplete datasets. To overcome this issue, we propose *missRows*, an *R* package that implements the MI-MFA method published in Voillet *et al.* (2016) [1], a multiple imputation (MI) approach in a multiple factor analysis (MFA) framework. The MI-MFA method generates multiple imputed datasets from a MFA model, then the results yield are combined in a single consensus solution. The package provides functions for estimating coordinates of individuals and variables in presence of missing individuals (rows), graphical outputs to display the results and improve interpretation, and various diagnostic plots to inspect the pattern of missingness and visualize the uncertainty due to missing row values. This vignette explains the use of the package and demonstrates typical workflows.

Contents

1 Introduction

Due to the increase in available data information, integrating large amounts of heterogeneous data is currently one of the major challenges in systems biology. Biological data integration provides scientists with a deeper insight into complex biological processes. However, when dealing with multiple data tables, the presence of missing values is a common situation for a variety of reasons. In omics data integration studies, it is common for some individuals to not be present in all data tables, resulting in a specific missing data pattern for multiple tables. Missing row values for a table of variables are challenging to handle because most statistical methods cannot be directly applied to incomplete datasets. To overcome one of the major issues associated with multiple omics data tables, we propose *missRows*, an *R* package that implements the MI-MFA method published in [1], a multiple imputation (MI) approach in a multiple factor analysis (MFA, [2]) framework. Proposed by Rubin (1987) [3], MI estimates both the parameters of interest and their variability in a data missingness framework, and it relies on the principle that a single value cannot reflect the uncertainty of the estimation of a missing value.

Figure 1 illustrates the three main steps in MI-MFA: imputation, analysis and combination. *missRows* stores the results of each step in a `S4` class: *MIDTList*. The analysis starts with observed and incomplete data tables $\boldsymbol{K}$. First, MI is used to generate plausible synthetic data values, called imputations, for missing values in the data. This step results in a number (M) of imputed datasets in which the missing data are replaced by random draws of plausible values according to a specific statistical model. The second step consists in analyzing each imputed dataset using MFA to estimate the parameters of interest. This step results in M analyses (instead of just one) which differ only because the imputations differ. Finally, MI combines all the results together to obtain a single consensus estimate, thereby combining varia-

tion within and across the M imputed datasets. In *missRows*, these three steps are performed using the function `MIMFA`, and are described in detail in our publication (Voillet *et al.* 2016, [1]).

The package provides functions for data exploration, and result visualization. The structure of missing values can be explored using the `missPattern` function. It gives to the user an indication of how much information is missing and how the missingness is distributed. The `plotInd` and `plotVar` functions provide scatter plots for individual and variable representations respectively.

In the MI-MFA framework, after estimating the configurations from the imputed datasets, a new source of variability due to missing values can be taken into account. The *missRows* package proposes two approaches to visualize the uncertainty of the estimated MFA configurations attributable to missing row values: confidence ellipses and convex hulls. The function `plotInd` contains methods implementing these two approaches.

This vignette explains the basics of using *missRows* by showing an example, including advanced material for fine tuning some options. The vignette also includes description of the methods behind the package.

Figure 1: Overview of the MI-MFA approach to handling missing rows in multi-omics data integration. The top part of the graphic indicates that analysis starts with observed, incomplete data tables K
In a first step, multiple imputation is performed using the hot-deck imputation approach: M imputed versions $\boldsymbol{K}^{(1)}, \ldots, \boldsymbol{K}^{(M)}$ of $\boldsymbol{K}$ are obtained by replacing the missing values by plausible data values. These plausible values are drawn from donor pools. The imputed sets are identical for the non-missing data entries, but differ in the imputed values. The second step is to estimate the configuration matrix $\boldsymbol{F}_m$ for each imputed dataset $\boldsymbol{K}^{(m)}$ using MFA. The estimated configurations differ from each other because their input data differ. The last step is to combine the M estimated configurations $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_M$ into a compromise configuration $\boldsymbol{F}_c$ using the STATIS method.

1.1 Citing *missRows*

We hope that *missRows* will be useful for your research. Please use the following information to cite *missRows* and the overall approach when you publish results obtained using this package, as such citation is the main means by which the authors receive credit for their work. Thank you!

González I., Voillet V. (2021). missRows: Handling Missing Individuals in Multi-Omics Data Integration. R package version 1.14.0.

Voillet V., Besse P., Liaubet L., San Cristobal M., González I. (2016). Handling missing rows in multi-omics data integration: Multiple Imputation in Multiple Factor Analysis framework. *BMC Bioinformatics*, 17(40).

1.2 How to get help for *missRows*

Most questions about individual functions will hopefully be answered by the documentation. To get more information on any specific named function, for example `MIMFA`, you can bring up the documentation by typing at the *R* prompt

```
help("MIMFA")
```

or

```
?MIMFA
```

The authors of *missRows* always appreciate receiving reports of bugs in the package functions or in the documentation. The same goes for well-considered suggestions for improvements. If you've run into a question that isn't addressed by the documentation, or you've found a conflict between the documentation and what the software does, then there is an active community that can offer help. Send your questions or problems concerning *missRows* to the Bioconductor support site at https://support.bioconductor.org.

Please send requests for general assistance and advice to the support site, rather than to the individual authors. It is particularly critical that you provide a small reproducible example and your session information so package developers can track down the source of the error. Users posting to the support site for the first time will find it helpful to read the posting guide at http://www.bioconductor.org/help/support/posting-guide.

1.3 Quick start

A typical MI-MFA session can be divided into three steps:

1. *Data preparation:* In this first step, a convenient *R* object of class *MIDTList* is created containing all the information required for the two remaining steps. The user needs to provide the data tables with missing rows, an indicator vector giving the stratum for each individual and optionally the name for each table.

2. *Performing MI:* Using the object created in the first step the user can perform MI-MFA to estimate the coordinates of individuals and variables on the MFA components, and the imputation of missing data values.

3. *Analysis of the results:* The results obtained in the second step are analyzed using visualization tools.

An analysis might look like the following. Here we assume there are two data tables with missing rows, `table1` and `table2`, and the stratum for each individual is stored in a data frame `df`.

```
## Data preparation
midt <- MIDTList(table1, table2, colData=df)

## Performing MI
midt <- MIMFA(midt, ncomp=2, M=30)

## Analysis of the results - Visualization
plotInd(midt)
plotVar(midt)
```

2 Using *missRows*

2.1 Installation

We assume that the user has the *R* program (see the *R* project at http://www.r-project.org) already installed.

The *missRows* package is available from the *Bioconductor* repository at http://www.bioconductor.org. To be able to install the package one needs first to install the core *Bioconductor* packages. If you have already installed *Bioconductor* packages on your system then you can skip the two lines below.

```
if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
BiocManager::install()
```

Warning: try http:// if https:// URLs are not supported

Once the core *Bioconductor* packages are installed, you can install the *missRows* package by

```
if (!requireNamespace("BiocManager", quietly=TRUE))
    install.packages("BiocManager")
BiocManager::install("missRows")
```

Warning: try http:// if https:// URLs are not supported

Load the *missRows* package in your *R* session:

```
library(missRows)
```

A list of all accessible vignettes and methods is available with the following command:

```
help.search("missRows")
```

2.2 Data overview

To help demonstrate the functionality of *missRows*, the package includes example datasets of the NCI-60 cancer cell line panel. This panel was developed as part of the Developmental Therapeutics Program (http://dtp.nci.nih.gov/) of the National Cancer Institute (NCI).

Once the package is loaded, these datasets can be used by typing

```
data(NCI60)
```

`NCI60` is a list composed of two components: `dataTables`, and `mae`.

```
names(NCI60)
```

```
[1] "dataTables" "mae"
```

`dataTables` contains both transcriptomic and proteomic expression from NCI-60 without missing data. The transcriptome data was directly retrieved from the `NCI60_4arrays` data in the *omicade4* package [4]. This data table contains gene expression profiles generated by the Agilent platform with only few hundreds of genes randomly selected to keep the size of the Bioconductor package small. The proteomic data table was retrieved from the *rcellminerData* package [5]. Protein abundance levels are available for 162 proteins. Both full datasets are accessible at https://discover.nci.nih.gov/cellminer/ [6].

Subsequently, a specific pattern of missingness has been created in these data tables for illustration purpose.

The component `dataTables$cell.line` contains the character vector of cancer type for each cell line: colon (CO), renal (RE), ovarian (OV), breast (BR), prostate (PR), lung (LC), central nervous system (CNS), leukemia (LE) and melanoma (ME).

```
table(NCI60$dataTables$cell.line$type)
```

```
 BR CNS  CO  LC  LE  ME  OV  PR  RE
  5   6   7   9   6  10   7   2   8
```

`NCI60$mae` contain a 'MultiAssayExperiment' instance from NCI60 data with transcriptome and proteomic experiments as described in `dataTables`.

```
NCI60$mae
```

```
A MultiAssayExperiment object of 2 listed
```

```
 experiments with user-defined names and respective classes.
 Containing an ExperimentList class object of length 2:
 [1] trans: data.frame with 300 rows and 48 columns
 [2] prote: ExpressionSet with 162 rows and 52 columns
Functionality:
 experiments() - obtain the ExperimentList instance
 colData() - the primary/phenotype DataFrame
 sampleMap() - the sample coordination DataFrame
 `$`, `[`, `[[` - extract colData columns, subset, or experiment
 *Format() - convert into a long or wide DataFrame
 assays() - convert ExperimentList to a SimpleList of matrices
 exportClass() - save data to flat files
```

2.3 Data preparation: the *MIDTList* object

The first step in using the *missRows* package is to create a *MIDTList* object. This object stores the data tables, the intermediate estimated quantities during the multiple imputation procedure and is the input of the visualization functions. The object *MIDTList* will usually be represented in the code here as `midt`.

To build such an object the user needs the following:

the incomplete data: data tables with missing rows. Two or more objects which can be interpreted as matrices (or data frames). Data tables passed as arguments must be arranged in samples $\times$ variables, with sample order and row names matching in all data tables.

strata: named vector or data frame giving the stratum for each individual. Names (in vector) or row names (in data frame) must be equal to the row names among data tables.

table names: optionally a character vector giving the name for each table.

Here, we demonstrate how to construct a *MIDTList* object from `NCI60` data. We first load the data

```
data(NCI60)
```

If the data tables are already available as a list, `tableList` say, and `cell.line` contains the character vector of cancer type for each cell line, then a *MIDTList* object can be created by

```
tableList <- NCI60$dataTables[1:2]
cell.line <- NCI60$dataTables$cell.line
```

```
midt <- MIDTList(tableList, colData=cell.line,
                    assayNames=c("trans", "prote"))
```

The function `MIDTList` makes a *MIDTList* object from separated data tables directly by

```
table1 <- NCI60$dataTables$trans
table2 <- NCI60$dataTables$prote
cell.line <- NCI60$dataTables$cell.line

midt <- MIDTList(table1, table2, colData=cell.line,
                    assayNames=c("trans", "prote"))
```

or by

```
midt <- MIDTList("trans" = table1, "prote" = table2,
                 colData=cell.line)
```

The function `MIDTList` makes a *MIDTList* object from 'MultiAssayExperiment' by

```
midt <- MIDTList(NCI60$mae)
```

A summary of the `midt` object can be seen by typing the object name at the *R* prompt

```
midt
```

```
An object of class MIDTList.

Tables:
      features individuals miss.indv
trans      300          48        12
prote      162          52         8

Strata:
 BR CNS  CO  LC  LE  ME  OV  PR  RE
  5   6   7   9   6  10   7   2   8
```

2.4 Performing MI-MFA

The main function for performing MI-MFA is `MIMFA`, and it has four main arguments. The first argument is an instance of class *MIDTList*. The second and third arguments are of type integer; they specify the number of components to include in MFA and

the number of imputations in MI, respectively. The fourth argument is a boolean, if `TRUE` then the number of components is estimated for data imputation, in this case the maximum number of components is determined by the second argument `ncomp`.

We then performs MI-MFA on the incomplete data tables contained in `midt` by using `M=30` imputations

```
midt <- MIMFA(midt, ncomp=50, M=30, estimeNC=TRUE)
```

`MIMFA` returns an object of class *MIDTList*. A short summary of this object is shown by typing

```
midt
```

```
An object of class MIDTList.

Tables:
      features individuals miss.indv
trans      300          48        12
prote      162          52         8

Strata:
 BR CNS  CO  LC  LE  ME  OV  PR  RE
  5   6   7   9   6  10   7   2   8

Multiple imputation in MFA
--------------------------
Total number of possible imputations: 2.09106e+15
Number of multiple imputations: 30
Estimated number of components for data imputation: 14
```

2.5 Working with the *MIDTList* object

Once the *MIDTList* object is created the user can use the methods defined for this class to access the information encapsulated in the object.

By example, the table names information is accessed by

```
names(midt)
```

```
[1] "trans" "prote"
```

For accessing the `colData` slot

```
cell.line <- colData(midt)
```

Multiple imputation calculates `M` configurations associated to each imputed datasets. In order to get the `M`th configuration, use the `configurations` function. For `M=5`,

```
conf <- configurations(midt, M=5)
dim(conf)
```

```
[1] 60 50
```

The list of ... can be exported by the function `function`

2.6 Data exploration and visualization

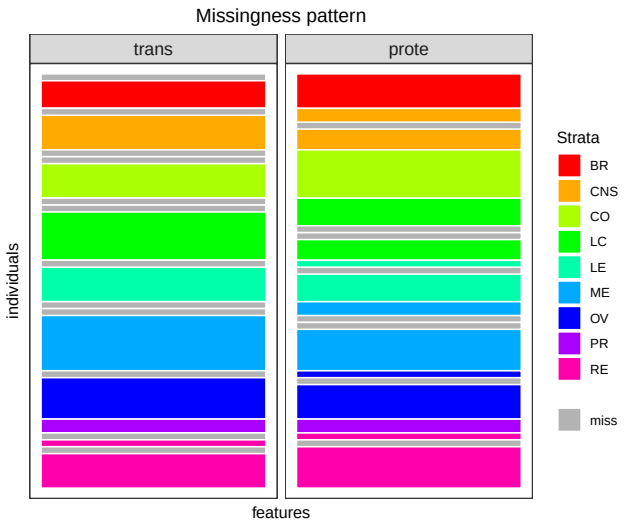
This section presents the available tools for data exploration and visualization of the results in *missRows*. Both `MIDTList` and `MIMFA` functions return an object of type *MIDTList*, and all of the following functions work with this object.

2.6.1 Inspect the missing rows pattern

Before imputation, the structure of missing values can be explored using visualization tools and summaries outputs. Looking at the missing data pattern is always useful. It can give you an indication of how much information is missing and how the missingness is distributed.

Inspect the missingness pattern for the `NCI60` incomplete data by

```
patt <- missPattern(midt, colMissing="grey70")
```



`missPattern` calculates the amount of missing/available rows in each stratum per data table and plots a missingness map showing where missingness occurs. Data tables are plotted separately on a same device showing the pattern of missingness. The individuals are colored with rapport to their stratum whereas missing rows are colored according to the `colMissing` argument.

The object `patt` encapsulates information from the structure of missingness, such as the amount of missing/available rows in each stratum per data table and the indicator matrix for the missing rows.

```
patt
```

```
Number of missing/available individuals in each stratum per data table.
Tables:
    trans prote
BR      1     0    1
CNS     1     1    2
CO      2     0    2
LC      2     2    4
LE      1     1    2
ME      2     2    4
OV      1     1    2
PR      0     0    0
RE      2     1    3
       12     8   20
```

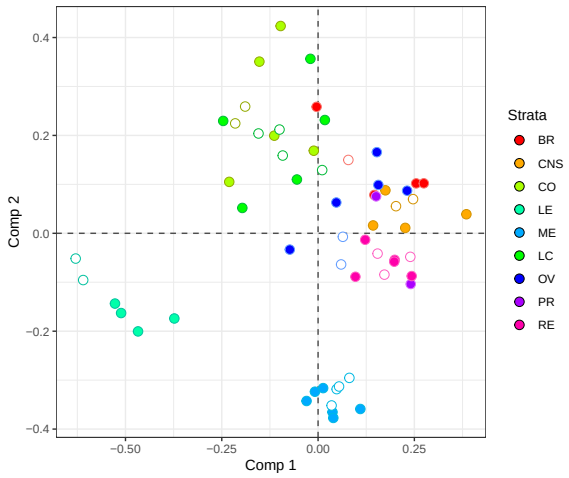
The missingness pattern shows that there are 20 missing rows in total: 12 for the `trans` table and 8 for the `prote` table. Moreover, there are two strata (`LC` and `ME`) with 4 missing rows, four strata with 2 missing rows, one stratum with 3 missing rows and another one with 1 missing row.

2.6.2 Individuals plot

An insightful way of looking at the results of `MIMFA` is to investigate how the observed and imputed individuals are distributed on the two-dimensional configuration defined by the MFA components.

`plotInd` function makes scatter plot for individuals representation from `MIMFA` results. Each point corresponds to an individual. The color of each individual reflects their corresponding stratum, whereas imputed individuals are colored according to the `colMissing` argument.

```
plotInd(midt, colMissing="white")
```



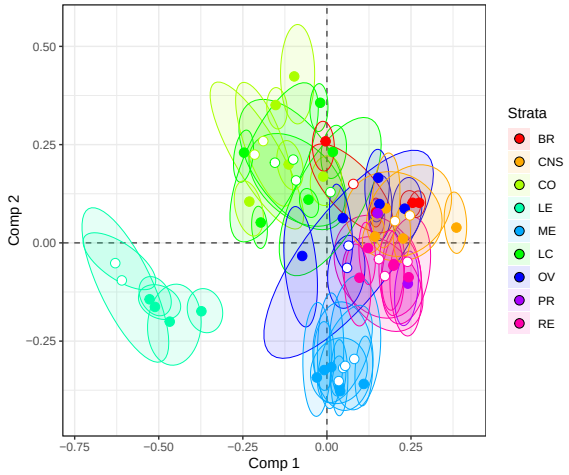
2.6.3 Visualizing the uncertainty induced by the missing individuals

Multiple imputation generates `M` imputed datasets and the variance between-imputations reflects the uncertainty associated to the estimation of the missing row values. The *missRows* package proposes two approaches to visualize and assess the uncertainty due to missing data: confidence ellipses and convex hulls.

The rough idea is to project all the multiple imputed datasets onto the compromise configuration. Each individual is represented by `M` points, each corresponding to one of the `M` configurations. Confidence ellipses and convex hulls can be then constructed for the `M` configurations for each individual. The computed convex hull results in a polygon containing all `M` solutions.

Confidence ellipses can be created using the function `plotInd` by setting the `confAreas` argument to `"ellipse"`

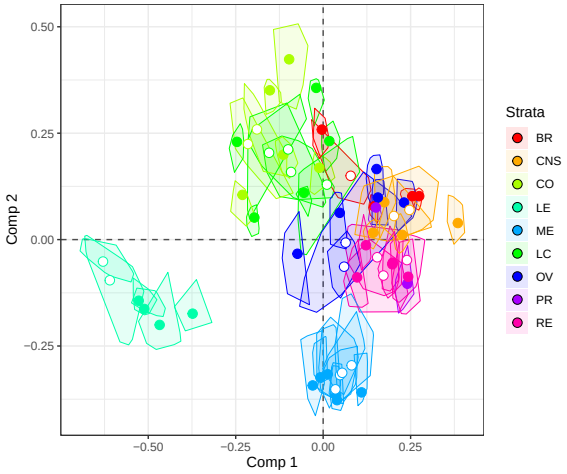
```
plotInd(midt, confAreas="ellipse", confLevel=0.95)
```



The 95% confidence ellipses show the uncertainty for each individual. For ease of understanding, not all individuals for the `M` configurations obtained are plotted.

Convex hulls are plotted by setting the `confAreas` argument to `"convex.hull"`

```
plotInd(midt, confAreas="convex.hull")
```

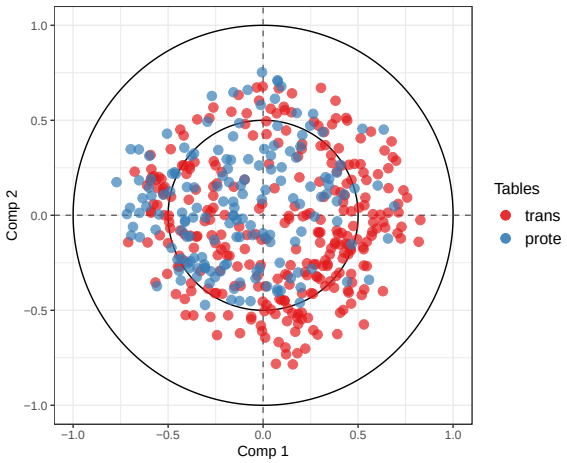


These graphical representations provide scientists with considerable guidance when interpreting the significance of `MIMFA` results. The larger the area of an ellipse (convex hull), the more uncertain the exact location of the individual. Thus, when ellipse and convex hull areas are large, the scientist should remain really careful regarding its interpretation.

2.6.4 Variables plot: correlation circle

`plotVar` produces a "correlation circle", *i.e.* the correlations between each variable and the selected components are plotted as scatter plot, with concentric circles of radius one and radius given by `radIn`. Variables are represented by points (symbols) and colored according to the data table to which they belong.

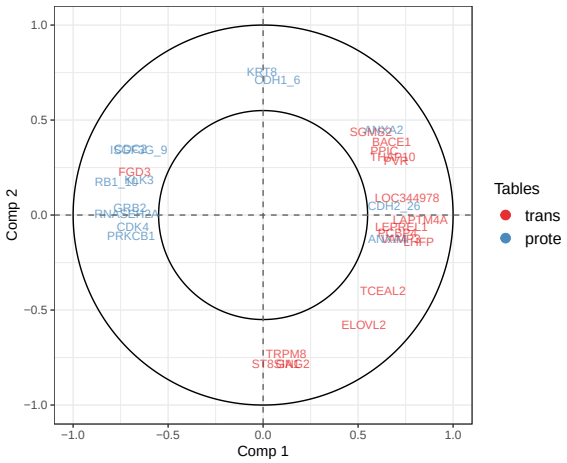
```
plotVar(midt, radIn=0.5)
```



This plot is an enlightening tool for data interpretation, as it enables a graphical examination of the relationships between variables. The variables or groups of variables strongly positively correlated are projected closely to each other on the correlation circle. When the correlation is strongly negative, the groups of variables are projected at diametrically opposite places on the correlation circle. The variables or groups of variables that are not correlated are situated 90° one from the other in the circle. For variables closely located to the origin, it means that some information can be carried on other axes and, it might be necessary to visualize the correlation circle plot in the subsequent dimensions. More details about the correlation circle interpretation can be found in [7].

In the high dimensional case, the interpretation of the correlation structure between variables from two or more data tables can be difficult, and a `cutoff` can be chosen to remove some weaker associations.

```
plotVar(midt, radIn=0.55, varNames=TRUE, cutoff=0.55)
```



2.7 How many imputations ?

When using MI, one of the uncertainties concerns the number `M` of imputed datasets needed to obtain satisfactory results. The number of imputed datasets in MI depends to a large extent on the proportion of missing data. The greater the missingness, the larger the number of imputations needed to obtain stable results. However, in multiple hot-deck imputation, the number of imputed datasets is limited by the size of the donor pools.

The appropriate number of imputations can be informally determined by carrying out MI-MFA on N replicate sets of M_l imputations for $l = 0, 1, 2, \ldots$, with $M_0 < M_1 < M_2 < \cdots < M_{max}$, until the estimate compromise configurations are stabilized.

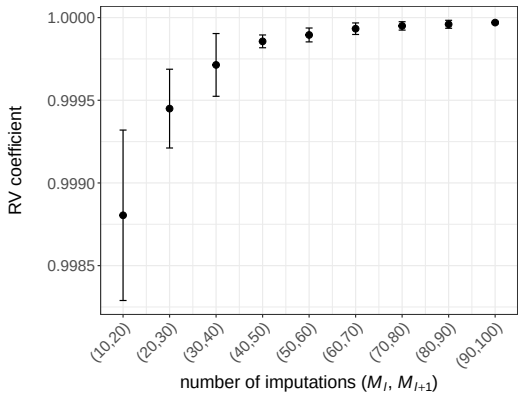
`tuneM` function implements such a procedure. Collections of size `N` are generated for each number of imputations `M`, with `M = seq(inc, Mmax, by=inc)`. The stability of the estimated MI-MFA configurations is then determined by calculating the RV coefficient between the configurations obtained using M_l and M_{l+1} imputations.

```
tune <- tuneM(midt, ncomp=2, Mmax=100, inc=10, N=20)
tune
```

```
tune$stats
```

```
  imputations       avg           sd
1     (10,20) 0.9988044 5.153783e-04
2     (20,30) 0.9994496 2.382201e-04
3     (30,40) 0.9997139 1.897539e-04
4     (40,50) 0.9998563 3.867360e-05
5     (50,60) 0.9998950 4.197550e-05
6     (60,70) 0.9999329 3.490001e-05
7     (70,80) 0.9999498 2.540659e-05
8     (80,90) 0.9999596 2.407006e-05
9    (90,100) 0.9999698 7.743291e-06
```

```
Warning:  Use of ‘df$x‘ is discouraged.  Use ‘x‘ instead.

Warning:  Use of ‘df$avg‘ is discouraged.  Use ‘avg‘ instead.

Warning:  Use of ‘df$avg‘ is discouraged.  Use ‘avg‘ instead.

Warning:  Use of ‘df$sd‘ is discouraged.  Use ‘sd‘ instead.

Warning:  Use of ‘df$avg‘ is discouraged.  Use ‘avg‘ instead.

Warning:  Use of ‘df$sd‘ is discouraged.  Use ‘sd‘ instead.

Warning:  Use of ‘df$x‘ is discouraged.  Use ‘x‘ instead.

Warning:  Use of ‘df$avg‘ is discouraged.  Use ‘avg‘ instead.
```



The values shown are the mean RV coefficients for the `N=20` two-dimensional configurations as a function of the number of imputations. Error bars represent the standard deviation of the RV coefficients.

3 Methods behind *missRows*

3.1 The MI-MFA approach

To deal with multiple tables with missing individuals, *missRows* implements the MI-MFA method in [1], a multiple imputation approach in a multiple factor analysis framework. According to MI methodology, the MI-MFA proceeds as follows. Let $\boldsymbol{K} = \{\boldsymbol{K}_1, \ldots, \boldsymbol{K}_J\}$ a set of J data tables, where each $\boldsymbol{K}_j$ corresponds to a table of quantitative variables measured on the same individuals, then carry out the following three steps:

1. Imputation: generate M different imputed datasets $\boldsymbol{K}^{(1)}, \ldots, \boldsymbol{K}^{(m)}, \ldots, \boldsymbol{K}^{(M)}$ of $\boldsymbol{K}$ using multiple hot-deck imputation [8].
2. MFA analysis: perform an MFA on each $\boldsymbol{K}^{(m)}$ imputed dataset leading to M different configurations $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_m, \ldots, \boldsymbol{F}_M$.
3. Combination: find a consensus configuration between all $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_M$ configurations using the STATIS method [9].

These steps are outlined in Figure 1 and described in detail in our publication (Voillet *et al.* 2016, [1]).

3.2 Imputation of missing rows and estimation of MFA axes

Even if the objective of MI-MFA is to estimate the MFA components in spite of missing values, an estimation of MFA axes and imputation of missing data values can be achieved. Consequently, the MI-MFA approach can also be viewed as a single imputation method. Moreover, since the imputation is based on a MFA model (on the components and axes), it takes into account similarities between individuals and relationships between variables.

Since the core of MFA is a PCA of the weighted data table $\boldsymbol{K}$, the algorithm suggested to estimate MFA axes and impute missing values is inspired from the alternating least squares algorithm used in PCA. This consists in finding matrices $\boldsymbol{F}$ and $\boldsymbol{U}$ which minimize the following criterion:

$$||\boldsymbol{K} - \boldsymbol{M} - \boldsymbol{F}\boldsymbol{U}||^2 = \sum_i \sum_k \left(\boldsymbol{K}_{ik} - \boldsymbol{M}_{ik} - \sum_{d=1}^{D} \boldsymbol{F}_{id}\boldsymbol{U}_{kd} \right)^2,$$

where $\boldsymbol{M}$ is a matrix with each row equal to a vector of the mean of each variable and D is the kept dimensions in PCA. The solution is obtained by alternating two multiple regressions until convergence, one for estimating axes (loadings $\hat{\boldsymbol{U}}$) and one for components (scores $\hat{\boldsymbol{F}}$):

$$\hat{\boldsymbol{U}}' = (\hat{\boldsymbol{F}}'\hat{\boldsymbol{F}})^{-1}\hat{\boldsymbol{F}}'(\boldsymbol{K} - \hat{\boldsymbol{M}})$$

$$\hat{\boldsymbol{F}} = (\boldsymbol{K} - \hat{\boldsymbol{M}})\hat{\boldsymbol{U}}(\hat{\boldsymbol{U}}'\hat{\boldsymbol{U}})^{-1}.$$

The algorithm to estimate MFA axes and impute missing values works as follows. Let $\boldsymbol{K} = [\boldsymbol{K}_1, \ldots, \boldsymbol{K}_J]$ the merged matrix containing missing values and $\boldsymbol{F}$ the matrix containing the compromise components of the MI-MFA, then carry out the following steps:

Step 0. Initialization $l = 0$: $\boldsymbol{K}^0$ is obtained by substituting missing values with initial values (for example with column means on the non-missing entries or zeros); $\hat{\boldsymbol{M}}^0$ is computed.

Step 1. Calculate $(\hat{\boldsymbol{U}}^l)' = (\boldsymbol{F}'\boldsymbol{F})^{-1}\boldsymbol{F}'(\boldsymbol{K}^{(l-1)} - \hat{\boldsymbol{M}}^{(l-1)})$.

Step 2. Missing values are estimated as $\hat{\boldsymbol{K}}^l = \hat{\boldsymbol{M}}^{(l-1)} + \boldsymbol{F}(\hat{\boldsymbol{U}}^l)'$.

Step 3. The new imputed data set $\boldsymbol{K}^l$ is obtained by replacing the missing values of the original K matrix with the corresponding elements of $\hat{\boldsymbol{K}}^l$, whilst keeping the observed values unaltered.

Step 4. $\hat{\boldsymbol{M}}^l$ is computed on $\boldsymbol{K}^l$.

Step 5. Steps 1 to 4 are repeated until convergence.

4 Session information

The following is the session info that generated this tutorial:

```
R version 4.1.1 (2021-08-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.3 LTS

Matrix products: default
BLAS:   /home/biocbuild/bbs-3.14-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.14-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_GB              LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] missRows_1.14.0             MultiAssayExperiment_1.20.0
 [3] SummarizedExperiment_1.24.0 Biobase_2.54.0
 [5] GenomicRanges_1.46.0        GenomeInfoDb_1.30.0
 [7] IRanges_2.28.0              S4Vectors_0.32.0
 [9] BiocGenerics_0.40.0         MatrixGenerics_1.6.0
[11] matrixStats_0.61.0          ggplot2_3.3.5

loaded via a namespace (and not attached):
 [1] gtools_3.9.2           tidyselect_1.1.1       xfun_0.27
 [4] purrr_0.3.4            lattice_0.20-45        colorspace_2.0-2
 [7] vctrs_0.3.8            generics_0.1.1         htmltools_0.5.2
[10] yaml_2.2.1             utf8_1.2.2             rlang_0.4.12
[13] pillar_1.6.4           glue_1.4.2             withr_2.4.2
[16] DBI_1.1.1              plyr_1.8.6             GenomeInfoDbData_1.2.7
[19] lifecycle_1.0.1        stringr_1.4.0          zlibbioc_1.40.0
[22] munsell_0.5.0          gtable_0.3.0           evaluate_0.14
[25] labeling_0.4.2         knitr_1.36             fastmap_1.1.0
[28] fansi_0.5.0            highr_0.9              Rcpp_1.0.7
[31] scales_1.1.1           BiocManager_1.30.16    DelayedArray_0.20.0
[34] XVector_0.34.0         farver_2.1.0           BiocStyle_2.22.0
```

```
       [37] digest_0.6.28          stringi_1.7.5          dplyr_1.0.7
       [40] grid_4.1.1             tools_4.1.1            bitops_1.0-7
       [43] magrittr_2.0.1         RCurl_1.98-1.5         tibble_3.1.5
       [46] crayon_1.4.1           pkgconfig_2.0.3        Matrix_1.3-4
       [49] ellipsis_0.3.2         assertthat_0.2.1       rmarkdown_2.11
       [52] R6_2.5.1               compiler_4.1.1
```

References

[1] V. Voillet, P. Besse, L. Liaubet, M. San Cristobal, and I. González. Handling missing rows in multi-omics data integration: multiple imputation in multiple factor analysis framework. *BMC Bioinformatics*, 17(1):402, 2016.

[2] B. Escofier and J. Pagès. Multiple factor analysis (AFMULT package). *Computational Statistics & Data Analysis*, 18(1):121–140, 1994.

[3] D. B. Rubin. *Multiple Imputation for Non-Response in Surveys*. Wiley-Interscience, Hoboken, New Jersey, USA, 2004.

[4] C. Meng, B. Kuster, A. Culhane, and A. M. Gholami. A multivariate approach to the integration of multi-omics datasets. *BMC Bioinformatics*, 2013.

[5] A. Luna, V. N. Rajapakse, F. G. Sousa, J. Gao, N. Schultz, S. Varma, W. Reinhold, C. Sander, and Y. Pommier. rcellminer: exploring molecular profiles and drug response of the NCI-60 cell lines in R. *Bioinformatics*, 2015.

[6] W. C. Reinhold, M. Sunshine, H. Liu, S. Varma, K. W. Kohn, J. Morris, J. Doroshow, and Y. Pommier. CellMiner: a web-based suite of genomic and pharmacologic tools to explore transcript and drug patterns in the NCI-60 cell line set. *Cancer Research*, 72(14):3499–3511, 2012.

[7] I. González, K. A. Lê Cao, M. J. Davis, and S. Déjean. Visualising associations between paired 'omics' data sets. *BioData Mining*, 5(1), 2012.

[8] G. Kalton and D. Kasprzyk. The treatment of missing survey data. *Survey Methodology*, 12:1–16, 1986.

[9] C. Lavit, Y. Escoufier, R. Sabatier, and P Traissac. The ACT (STATIS method). *Computational Statistics & Data Analysis*, 18(1):97–119, 1994.