Deliverable 4.1

R Programmes for Robust Procedures Including Manual

Version: 2011

Beat Hulliger, Andreas Alfons, Peter Filzmoser, Angelika Meraner, Tobias Schoch, Matthias Templ

The project FP7–SSH–2007–217322 AMELI is supported by European Commission funding from the Seventh Framework Programme for Research.

http://ameli.surveystatistics.net/
Contributors of Deliverable 4.1

Chapter 1: Beat Hulliger and Tobias Schoch, University of Applied Sciences Northwestern Switzerland; Andreas Alfons, Peter Filzmoser, Angelika Meraner and Matthias Templ, Vienna University of Technology

Chapter 2: Beat Hulliger and Tobias Schoch, University of Applied Sciences Northwestern Switzerland

Chapter 3: Andreas Alfons, Peter Filzmoser, Angelika Meraner and Matthias Templ, Vienna University of Technology

Chapter 4: Beat Hulliger and Tobias Schoch, University of Applied Sciences Northwestern Switzerland

Chapter 5: Beat Hulliger and Tobias Schoch, University of Applied Sciences Northwestern Switzerland

Main Responsibility

Beat Hulliger, University of Applied Sciences Northwestern Switzerland

Evaluators

Internal experts: Ralf Münnich, Christian Bruch, Tobias Enderle, Jan-Philipp Kolb and Stefan Zins, University of Trier
Aim and Objectives of Deliverable 4.1

Indicators and in particular the Laeken indicators are vulnerable to outliers. Outliers may not only bias the indicators but also introduce a high additional variability. Therefore, outliers and other deviations from theoretical distributions may undermine the quality of indicators thoroughly. Robust procedures remain stable in those situations but are more complex to handle.

In the workpackage on robustness (WP 4), we have developed robust imputation procedures, in particular for multivariate data, including detection of outlying and influential observations and small area estimation procedures. Further, we have developed robust (non- and semi-parametric) estimators for important Laeken indicators. The methodological details of the robust estimators/procedures are documented in the Deliverable 4.2, see Hulliger et al. (2011b). These procedures were implemented in the statistical programming language R (R Development Core Team, 2011), are available in R-packages, and are documented in this Deliverable, D4.1. A draft internal version of D4.1 was established 26 March 2010 and circulated before the simulations started. This final version of D4.1 is a slight update of the first version.

The evaluation of the robustness of classical indicators, in particular the Laeken indicators, and of the new robust procedures are described in Deliverable D7.1; see Hulliger et al. (2011a). Quality measures of the robustness of indicators and of the impact of robust procedures were developed in Workpackage 4 and implemented in the Workpackages 6 and 7. As a result of the analysis carried out on the robustness of procedures in Workpackage 4, 6, and 7 recommendations are formulated in Deliverable D7.1 for the use of indicators in what concerns robustness issues.

Workpackage 4 was only possible due to an intensive and fruitful collaboration between Technical University of Vienna and University of Applied Sciences Northwestern Switzerland. The discussions on contaminations and the subsequent implementation in the simulation environments of the AMELI project have brought research on robustness in survey sampling to a new level. The insight that was possible due to the simulations and the new methods developed would not have been possible without the support of the European Union through the Socio-economic Sciences and Humanities programme of the 7th Framework Programme for Research and Development. The workpackage contributors are greatful for the support of the European Union and for the additional support granted from their respective institutions.
Contents

I  Manual  2

1  Introduction  3

2  Basic Functions  5

  msvymean  .................................................. 6
  mer  ......................................................... 9
  plot.mer  ................................................... 11
  msvyratio  .................................................. 12
  tsvymean  ................................................. 15
  rht-utils  ............................................... 17
  rht-internal  ............................................ 19
  asymhuberPsi  ........................................... 20
  asymhuberPsi  ........................................... 21

3  Robust Distribution Fitting  22

  fitPareto  .................................................. 23
  thetaPDC  .................................................. 25
  thetaWML  .................................................. 27

4  Robustification of the Quintile Share Ratio Estimator  29

  BQSR  ....................................................... 30
  MQSR  ....................................................... 32
5 Multivariate Robust Imputation

5.1 Mahalanobis-Distance Based Robust Imputation
   BEM
   GIMCD
   POEM
   TRC
   plotMD
   weighted.var

5.2 Depth Based Robust Imputation
   EA
   ER

5.3 Robust Imputation for Compositions
   aDist
   impCoda
   impKNNa
   invilr
   ilr
   robVariation

5.4 Sequential Robust Imputation
   irmi

II Code

6 RHT Package

7 Robust Distribution Fitting

8 Robustification of the QSR

9 MODI Package

10 Robust Imputation for Compositions

11 Sequential Robust Imputation
Part I

Manual
Chapter 1

Introduction

Workpackage 4 of the AMELI project has developed methods to deal with the estimation of complex population characteristics, the Laeken-indicators, for heavily asymmetric and zero-inflated distributions, here income and its components, when the data stems from complex surveys and contains outliers and missing values.

These methods have been implemented in R (R Development Core Team, 2011) and will be evaluated by the simulations in Workpackage 6 of the AMELI project. The present Deliverable D4.1 gives the technical background for the methods that should go into the simulation study. It uses the R documentation standard to introduce the packages and functions and it gives full documentation of the code. The code contains all comments, which helps later development, but it may be very long. While Part I of the Deliverable D4.1 is a manual, Part II is intended as a reference and documentation for programmers. Thus the user of D4.1 may print out Part I at his or her convenience but should carefully evaluate, whether Part II needs to be printed. For each section in Part I of D4.1 there is a corresponding section in Part II with the code of the functions.

The theoretical background and the exact description of the methods is presented in Deliverable D4.2 of Workpackage 4 of the AMELI project; see Hulliger et al. (2011b). The purpose of D4.1 is to fix and document the procedures that go into the simulation, i.e. to provide the necessary technical documentation. Deliverable D4.2 will be the methodological report that explains and discusses the methods in-depth.

Deliverable D4.1 is structured as follows: A series of robust methods for estimation of means and for ratio estimation has been implemented in package for Robustified Horvitz-Thompson Estimators rht. Estimation of means and totals is not directly needed for the Laeken indicators. However, these estimators are important, first because important macro-economic indicators like total social transfers in a country depend directly on the estimation of means. In addition, estimation of a mean is important for comparison with other methods. These estimators are described in Chapter 2.

A series of methods aims at fitting in a robust way a distribution to the observed incomes and then derives the corresponding Laeken indicator from the fitted distribution. These functions are described in Section 3.

The Quintile-share-ratio is the most vulnerable of the Laeken indicators because it is specifically designed to measure inequality. A series of robust methods which help to
find a compromise between robustness and bias are implemented in package \texttt{rqsr} and described in Section 4.

Methods for multivariate outlier detection and robust imputation are presented in Section 5. A first set of functions is implemented in the package \texttt{modi} (Sections 5.1 and 5.2) while special functions are developed for compositions, i.e. for the situation where the aggregate of income components is considered fixed but the components may contain outliers (Section 5.3). A robust sequential (model-based) imputation method is discussed in Section 5.4.
Chapter 2

Basic Functions

This chapter is dedicated to outlier-robust Horvitz-Thompson estimation of elementary functions in complex samples such as means and totals. These basic functions serve as building blocks in several Laeken indicator. On the other hand, estimation of a mean is important for comparison with other methods. These estimators are described in the chapter at hand. The code can be found in Chapter 6.
msvymean

Robust M-estimation of the mean for complex samples (robust Horvitz-Thompson or robust weighted mean)

Description

`msvymean` computes robust Horvitz-Thompson estimates of the mean or robust weighted mean estimates for complex samples, using M-estimation. This Package depends on Thomas Lumley’s `survey` package.

Usage

```r
## S3 method for class 'survey.design':
msvymean(y, design, type=c("rht", "rwm"), na.rm=T, k=3, steps=50,
plot=F, acc=1e-6, quietly=F, psi=huberPsi, exact=F)

## S3 method for class 'svystat.rob':
print(object)

## S3 method for class 'svystat.rob':
summary(object)

## S3 method for class 'plotable':
plot(object)
```

Arguments

- **y**: a formula object (only one variable)
- **design**: a `survey.design` object
- **type**: either `rht` for robust Horvitz-Thompson estimator (default), or `rwm` for robust weighted mean estimator
- **na.rm**: Should cases with missing values be dropped? (default `TRUE`)
- **k**: robustness tuning constant
- **steps**: maximal number of IRLS iterations (default: 50, i.e. usually fully iterated estimation)
- **plot**: Plot the residuals? (default: `FALSE`)
- **acc**: Numerical convergence criterion (default: `1e-6`)
- **quietly**: omit return values (default: `FALSE`)
- **psi**: psi function of the class `psi_func-class` in the `robustbase` package.
- **exact**: if `TRUE` variance estimates are computed considering the preliminary scale estimate (MAD) (`FALSE` by default; see Details below)
Details

msvymean performs (inverse probability-) weighted M-estimation (by default with huberPsi function), with each observation being weighted by the inverse of its sampling probability. The choice of using a robust Horvitz-Thompson estimator (type=rht) or robust weighted mean estimator (type=rwm) depends on the underlying survey design (If \( y \) is positively correlated with the inclusion probabilities a rht type estimator should be used, else rwm). You may set steps equal to one to get a one-step estimation. Variance estimates are computed as first-order linearization using the design-based estimation facilities in the survey package.

msvymean allows also the estimation for domains. Use the command subset and a design subset expression instead of the original survey.design object in msvymean (see examples for more details).

Note that there are useful rht-utility functions: summary, plot, coef, and vcov (See also examples).

In case of an asymmetric distribution, the user may choose an asymmetric Huber psi-function (or any other psi-function of the class psi_func-class in the robustbase package). This can be done by calling msvymean with asymhuberPsi as additional argument.

Users may set exact=TRUE to compute a linearization variance estimate considering that the MAD has been used as preliminary scale estimate. However, the estimates may become very unstable.

Value

Object of class svystat.rob, which is scalar with a var attribute giving the variance, a statistic attribute giving the name of the statistic, a k attribute giving the robustness tuning constant, and a method attribute indicating the computation method. Each M-estimation object has an attributed consisting of the psi function object. In addition the objects possess further attributes concerning number of observations, number of NA’s, number of declared outliers, average weight, and several details with regard to the optimization.

Author(s)

Beat Hulliger and Tobias Schoch

References

See Also

svymean

Examples

```r
## load "api" data set from "survey" package (a description of the data set
## can be found there)
data(api)
## define "survey.design" for stratified sampling
dstrat <- svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
## compute a robust Horvitz-Thompson estimate for the mean of the variable
## "api00" (Academic Performance Index in 2000)
rht1 <- msvymean(~api00, dstrat, k=1.2)
# get a summary of the estimation
summary(rht1)
## robust Horvitz-Thompson estimates for a domain of the variable. Here we are
## interested in the robust mean for "api00" for (sch.wide == "Yes"). That is
## the average of the academic performance in 2000 only for the schools that
## met the school-wide growth target.
msvymean(~api00, subset(dstrat, sch.wide == "Yes"), k=1.2)
## plot method
plot(rht1)
## to extract the estimate from the object
coef(rht1)
## to extract the variance from the object
vcov(rht1)
```
**mer**

*Minimum Estimated Risk M-Estimation*

**Description**

`mer` searches for the robustness tuning parameter $k$ (for M-estimation) that minimizes the (inverse-probability weighted) mse. Thus, MER-estimation is a strategy to adaptively choose optimal robustness tuning. (for L-estimation MER is not yet implemented)

**Usage**

```r
## S3 method for class 'svystat.rob':
mer(object, init = 0.1, box.lo = 1e-04, tol = 1e-04)
## S3 method for class 'mer':
summary(object)
```

**Arguments**

- `object` an object of the class `svystat.rob` (i.e. an estimate of `msvymean` with a first guess of the robustness tuning parameter $k$)
- `init` an initial value for the parameters to be optimized over, i.e. of the optimal $k$ (default 0.1)
- `box.lo` lower bound (box-constraint) on the variables for the *L-BFGS-B* method (default 1e-4)
- `tol` numerical tolerance criterion (delivered to the `.irls` function)

**Details**

`mer` searches for the robustness tuning parameter $k$ (for a M-estimator) that minimizes the (inverse-probability weighted) mean squared error (mse). The function `mer` calls `optim` (in the `stats` package) to search for an optimal tuning constant $k$ that minimizes the estimated risk (mer). Minimization is performed using the *L-BFGS-B* method (Byrd et al., 1995; Nocedal and Wright, 2006), i.e. a limited-memory modification of the BFGS quasi-Newton method. By default the following box-constraints are used: lower=1e-4, upper=inf. Note that in typical applications, neither the box-constraints nor the initial value for the parameters to be optimized over, need to be adapted. The algorithm usually converges in a couple of iterations, since it capitalizes (by means of a finite-difference approximation of the gradient) on the almost quadratic shape of the mse (at least for asymmetric distributions) w.r.t. the tuning constant.
Important notice: In case of symmetric distributions, mer-estimation tends to choose optimal tuning constants $k$ that are far too large. Sometimes the global minimum of the mse is at zero. In such a case, smaller $k$'s (i.e. downweighting a larger amount of observations) will always reduce the mse and the optimal M-estimator may be, e.g., the median.

Algorithm not converged: If the algorithm has not converged, set the initial value (i.e. `init`) of $k$ near the 'true' $k$. In addition, you may modify the numeric convergence criterion `tol`.

Note that there are useful utility functions: `summary`, `coef`, and `vcov` (See also `examples`). In addition, there is a plot method associated with `mer`, see `plot`.

Value

Object of the class(es) `svystat.rob` and `mer`.

Author(s)

Beat Hulliger and Tobias Schoch

References


Examples

# A simple example

m1 <- msvymean(~api00, dstrat, k=0.3)
m1.mer <- mer(m1)
summary(m1.mer)
plot(m1.mer)
Description

plot is a S3 method for objects of the class mer. It plots the mse, variance and mean of an mer-object and shows the minimum estimated risk by a vertical line in all plots.

Usage

## S3 method for class 'mer':
plot(object, n = 10, scale = 1)

Arguments

object an object of the class mer
n number of points at which the mse is evaluated (default 10)
scale specify the plot range (default: 1)

Details

plot is a S3 method for objects of the class mer. Specify the n argument to change the number of points at which the mse (and the variance and mean) are evaluated. Note that the number of evaluation points must be carefully chosen since the computation time for even a single evaluation can be extremely high. Specify the scale argument to define the plot range.

Author(s)

Beat Hulliger and Tobias Schoch

Examples

## three simple examples
m1 <- msvymean(~api00, dstrat, k=0.1)
t1 <- mer(m1)
## plot default
plot(t1)
## evaluate the function at 50 points
plot(t1, n=50)
## larger plot range
plot(t1, scale=2)
msvyratio  

Robust ratio M-estimation for complex samples

Description

msvyratio computes a robust ratio estimate for complex samples, using M-estimation. This Package depends on Thomas Lumley’s survey package.

Usage

```r
## S3 method for class 'survey.design'
msvyratio(numerator, denominator, design, na.rm = T, k = 3, steps = 50,
          plot = F, acc = 1e-06, quietly=FALSE)
## S3 method for class 'svystat.rob'
print(object)
## S3 method for class 'svystat.rob'
summary(object)
## S3 method for class 'plotable'
plot(object)
```

Arguments

- `numerator`: a formula object (only one variable)
- `denominator`: a formula object (only one variable)
- `design`: a survey.design object
- `na.rm`: Should cases with missing values be dropped? (default TRUE)
- `k`: robustness tuning constant
- `steps`: maximal number of IRLS iterations (default: 50, i.e. fully iterated estimation)
- `plot`: Plot the residuals? (default: FALSE)
- `acc`: Numerical convergence criterion (default: 1e-6)
- `quietly`: omit return values (default=FALSE)

Details

msvyratio computes a robust ratio estimate for complex samples, using M-estimation. Variance estimates are computed as first-order linearization using the design-based estimation facilities in the survey package.

You may set `steps` equal to one to obtain an one-step estimation.
msvyratio allows also the estimation for domains. Use the command subset and a design subset expression insted of the original survey.design object in msvyratio (see examples for more details).

Note that there are useful rht-utility functions: summary, plot, coef, and vcov (See also examples).

Value

Object of class svystat.rob, which is scalar with a var attribute giving the variance, a statistic attribute giving the name of the statistic, a k attribute giving the robustness tuning constant, and a method attribute indicating the computation method. In addition the objects possess further attributes concerning number of observations, number of NA’s, number of declared outliers, average weight, and several details with regard to the optimization.

Author(s)

Beat Hulliger and Tobias Schoch

References


See Also

svyratio

Examples

```r
## load "api" data set from "survey" package (a description of the data set
## can be found there)
data(api)
## define "survey.design" for stratified sampling
dstrat <- svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
## compute a robust Horvitz-Thompson estimate for the mean of the variable
## "api00" (Academic Performance Index in 2000)
ratio1 <- msvyratio(~api00, ~api99, dstrat, k=1.2)
# get a summary of the estimation
summary(ratio1)
## robust Horvitz-Thompson estimates for a domain of the variable. Here we are
## interested in the robust mean for "api00" in case of (sch.wide == "Yes").
## That is the average of the academic performance in 2000 only for the
```
## schools that met the school-wide growth target.
msvyratio(~api00, ~api99, subset(dstrat, sch.wide == "Yes"), k=1.2)

## plot method
plot(ratio1)

## to extract the estimate from the object
coef(ratio1)

## to extract the variance from the object
vcov(ratio1)
tsvymean  

Trimmed and winsorized weighted mean for complex samples

Description

`tsvymean` computes either the trimmed or winsorized weighted mean for complex samples. This Package depends on Thomas Lumley’s `survey` package.

Usage

```r
## S3 method for class 'survey.design':
svymean(y, design, trim=c(0, 1), type=c("trim", "win"), na.rm=T, 
quietly=FALSE)
## S3 method for class 'svystat.rob':
print(object)
## S3 method for class 'svystat.rob':
summary(object)
```

Arguments

- **y**: a formula object (only one variable)
- **design**: a `survey.design` object
- **trim**: Range of observations \([lo, hi]\) to be used in the computation of the weighted mean. The fraction \(lo\) of observations is trimmed from the lower end and the fraction \(1-hi\) is trimmed from the upper end (\(lo < hi\)).
- **type**: either `trim` for trimming (default), or `win` for winsorization
- **na.rm**: Should cases with missing values be dropped? (default `TRUE`)
- **quietly**: omit return values (default=`FALSE`)

Details

By default `trim` equals \(c(0,1)\) and the regular weighted mean is computed. The variance estimators are based on first-order linearizations using the design-bases estimation facilities of the `survey` package. For reasons of numerical stability, the variance of the winsorized weighted mean is computed using the variance estimator of the trimmed mean. The variance estimate of the winsorized weighted mean can be found in the `robustness` attributes of the `svystat.rob` object.

`tsvymean` allows also the estimation for domains. Use the command `subset` and a `design subset expression` insted of the original `survey.design` object in `tsvymean` (see examples for more details).
Note that there are useful rht-utility functions: `summary`, `plot`, `coef`, and `vcov` (See also examples).

**Value**

Object of class `svystat.rob`, which is scalar with a `var` attribute giving the variance, a `statistic` attribute giving the name of the statistic, a `k` attribute giving the robustness tuning constant, and a `method` attribute indicating the computation method. In addition the objects possess further attributes concerning number of observations, number of NA’s, number of declared outliers, average weight, and several details with regard to the optimization.

**Author(s)**

Beat Hulliger and Tobias Schoch

**References**


**See Also**

`svymean`

**Examples**

```r
## load "api" data set from "survey" package (a description of the data set
## can be found there)
data(api)
## define "survey.design" for stratified sampling
dstrat <- svydesign(id=~1,strata=~stype, weights=~pw, data=apistrat, fpc=~fpc)
## compute a robust Horvitz-Thompson estimate for the mean of the variable
## "api00" (Academic Performance Index in 2000)
tm1 <- tsvymean(~api00, dstrat, trim=c(0.01, 0.09), type="trim")
# get a summary of the estimation
summary(tm1)
## robust estimates for a domain of the variable. Here we are interested in
## the trimmed mean for "api00" in case of (sch.wide == "Yes"). That is the
## average of the academic performance in 2000 only for the schools that met
## the school-wide growth target.
tsvymean(~api00, subset(dstrat, sch.wide == "Yes"), trim=c(0.01, 0.09),
          type="trim")
## to extract the estimate from the object use
coef(tm1)
## to extract the variance from the object use
vcov(tm1)
```
Description

The \texttt{rht} package contains some useful utility functions to extract relevant information from objects or to plot or summarize the objects of the class \texttt{svystat.rob}.

Usage

\begin{verbatim}
## S3 method for class 'svystat.rob':
  summary(object)
## S3 method for class 'plotable':
  plot(object)
## S3 method for class 'svystat.rob':
  coef(object)
## S3 method for class 'svystat.rob':
  vcov(object)
## S3 method for class 'svystat.rob':
  print(object)
\end{verbatim}

Details

\texttt{summary} is a method to summarize the object  
\texttt{plot} is a plot method to display diagnostic plots for the functions \texttt{msvymean} and \texttt{msyvyratio}  
\texttt{coef} is a method to extract the estimates (coefficients) from a \texttt{svystat.rob} object  
\texttt{vcov} is a method to extract the variance from a \texttt{svystat.rob} object  
\texttt{print} is an (internal) print method

Author(s)

Beat Hulliger and Tobias Schoch

Examples

\begin{verbatim}
## load "api" data set from "survey" package (a description of the data set  
## can be found there)  
data(api)  
## define "survey.design" for stratified sampling  
dstrat <- svydesign(id=1,strata=stype, weights=pw, data=apistrat, fpc=fpc)
\end{verbatim}
## compute a robust Horvitz-Thompson estimate for the mean of the variable
## "api00" (Academic Performance Index in 2000)
rht1 <- msvymean(~api00, dstrat, k=4)
# get a summary of the estimation
summary(rht1)

## robust Horvitz-Thompson estimates for a domain of the variable. Here we
## are interested in the robust mean for "api00" in case of (sch.wide ==
## "Yes"). That is the average of the academic performance in 2000 only for the
## schools that met the school-wide growth target.
msvymean(~api00, subset(dstrat, sch.wide == "Yes"), k=4)
# plot method
plot(rht1)

## to extract the estimate from the object
coef(rht1)
## to extract the variance from the object
vcov(rht1)
Description

Internal functions (should not be called by the user) for the rht package. For user information ask for help about rht.
asymhuberPsi  

Asymmetric Huber Psi function

Description

The asymmetric Huber psi function is a slightly modified psi function for M-estimation for asymmetric data.

Details

asymhuberPsi is an asymmetric variant of the standard Huber psi function. It is an object of the psi_func-class class in the robustbase package.

Author(s)

Beat Hulliger and Tobias Schoch
asymhuberPsi  

(modified) Huber Psi function

Description

This Huber psi function is a slightly modified Huber psi function for M-estimation.

Details

huberPsi is an object of the psi_func-class class in the robustbase package. This function is a modified variant of the huberPsi function in the robustbase package with exception handling (can cope with NA, Inf, etc.).

Author(s)

Beat Hulliger and Tobias Schoch
Chapter 3

Robust Distribution Fitting

The code for this chapter can be found in Part II, Chapter 7.
fitPareto  

*Fit income distribution models with the Pareto distribution*

**Description**

Fit a Pareto distribution to the upper tail of the income. Since a theoretical distribution is used for the upper tail, this is a semi-parametric approach.

**Usage**

```r
fitPareto(x, k, method = "thetaPDC", group = NULL, ...)
```

**Arguments**

- `x`  
a numeric vector.

- `k`  
the number of observations in the upper tail to which the Pareto distribution is fitted.

- `method`  
either a function or a character string specifying the function to be used to estimate the shape parameter of the Pareto distribution, such as `thetaPDC` (the default). See “Details” for requirements for such a function and “See also” for available functions.

- `group`  
a vector or factor specifying groups of elements of `x` (e.g., households). If supplied, each group of observations is expected to have the same value in `x` (e.g., household income). Only the values of the first group member to appear are used for fitting the Pareto distribution. For each group above the threshold, every group member is assigned the same value.

- `...`  
additional arguments to be passed to the specified method.

**Details**

The function supplied to `method` should take a numeric vector (the observations) as its first argument. Its second argument should be the number of observations in the upper tail to which the Pareto distribution is fitted. Additional arguments are passed via the `...` argument.

**Value**

A numeric vector with a Pareto distribution fit to the upper tail.
Author(s)

Andreas Alfons and Josef Holzer

See Also

`thetaPDC, thetaWML, thetaHill, thetaISE, thetaLS, thetaMoment, thetaQQ, thetaTM`

Examples

data(eusilc)

# gini coefficient without Pareto tail modeling
gini("eqIncome", weights = "rb050", data = eusilc)

# gini coefficient with Pareto tail modeling
eqIncome <- fitPareto(eusilc$eqIncome,
  k = 75, group = eusilc$db030)
gini(eqIncome, weights = eusilc$rb050)
Description

The partial density component (PDC) estimator estimates the shape parameter of a Pareto distribution based on the relative excesses of observations above a certain threshold.

Usage

thetaPDC(x, k, ...)

Arguments

x a numeric vector.
k the number of observations in the upper tail to which the Pareto distribution is fitted.
... additional arguments to be passed to nlm (see “Details”).

Details

The PDC estimator is obtained by minimizing the integrated squared error (ISE) criterion with an incomplete density mixture model. The minimization is carried out using nlm. By default, the starting value is the Hill estimator.

Value

The estimated shape parameter.

Author(s)

Josef Holzer and Andreas Alfons

References

See Also

`fitPareto`, `thetaISE`, `thetaHill`

Examples

data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
thetaPDC(eusilc$eqIncome[!duplicated(eusilc$db030)], k = 75)
**thetaWML**  
*Weighted maximum likelihood estimator*

**Description**

Estimate the shape parameter of a Pareto distribution using a weighted maximum likelihood approach.

**Usage**

```r
thetaWML(x, k, weight = c("residuals", "probability"),
         const, bias = TRUE, tol = 1e-5, ...)
```

**Arguments**

- `x`: a numeric vector.
- `k`: the number of observations in the upper tail to which the Pareto distribution is fitted.
- `weight`: a character string specifying the weight function to be used. If `residuals` (the default), the weight function is based on standardized residuals. If `probability`, probability based weighting is used. Partial string matching allows these names to be abbreviated.
- `const`: Tuning constant(s) that control the robustness of the method. If `weight=residuals`, a single numeric value is required (the default is 2.5). If `weight=probability`, a numeric vector of length two must be supplied (a single numeric value is recycled; the default is 0.005 for both tuning parameters). See the reference for more details.
- `bias`: a logical indicating whether bias correction should be applied.
- `tol`: the desired accuracy (see Details).
- `...`: additional arguments to be passed to `nlm` (see Details).

**Details**

The weighted maximum likelihood estimator belongs to the class of M-estimators. In order to obtain the estimator, the root of a certain function needs to be found. This is implemented by minimizing the squared function with `nlm`. The Hill estimator is thereby the starting value for the minimization. Afterwards, it is checked whether the value of the aforementioned function for the obtained minimum lies within the specified tolerance `tol` from 0.
Value

The estimated shape parameter.

Author(s)

Josef Holzer and Andreas Alfons

References


See Also

*fitPareto*, *thetaHill*

Examples

data(eusilc)
# equivalized disposable income is equal for each household # member, therefore only one household member is taken thetaWML(eusilc$eqIncome[!duplicated(eusilc$db030)], k = 75)
Chapter 4

Robustification of the Quintile Share Ratio Estimator

The code for this chapter can be found in Part II, Chapter 8.
BQSR

Bias-compensated, Trimmed (robust) Quintile Share Ratio Estimator

Description

BQSR computes robust Horvitz-Thompson quintile share ratio estimates using trimming. This Package depends on the packages survey and rht.

Usage

BQSR(x, design, lower=0, upper=0)

Arguments

x a formula object (equivalized income)
design a survey.design object
lower parameter for compensation
upper trimming parameter

Details

BQSR performs (inverse probability-) weighted, bias-compensated trimmed quintile share ratio estimator based on the function tsvymean in the rht package. The influence of outlying or influential observations in the upper tail of the income distribution on the quintile share mean of the rich (i.e. numerator of the QSR) is reduced by means of trimming. On the other hand, the researcher may specify the compensation parameter to adjust the quintile share mean of the poor (i.e. denominator) in order to minimize the bias that may have been induced due to trimming. Variance estimates are computed as first-order linearization using the design-based estimation facilities in the survey package.

Value

Object of class svystat.rob, which is scalar with a var attribute giving the variance, a statistic attribute giving the name of the statistic, a k attribute giving the robustness tuning constant, and a method attribute indicating the computation method. In addition the objects possess further attributes concerning number of observations, number of NA’s, number of declared outliers, average weight, and several details with regard to the optimization.
Author(s)

Beat Hulliger and Tobias Schoch

References


See Also

svymean, tsvymean, msvymean, MQSR.

Examples

## Pseudo example
BQSR(eqIncome, datsilc2004, 0, 0.01)
MQSR

M-Estimation Quintile Share Ratio Estimator

Description

MQSR computes robust Horvitz-Thompson M-Estimation quintile share ratio. This Package depends on the packages `survey` and `rht`.

Usage

```r
BQSR((x, design, k=4)
```

Arguments

- `x`: a formula object (equivalized income)
- `design`: a `survey.design` object
- `k`: robustness tuning constant (see `huberPsi`)

Details

BQSR performs (inverse probability-) weighted M-Estimation quintile share ratio estimator based on the function `msvymean` in the `rht` package. That is, the influence of outlying and influential observations in the upper tail of the income distribution is reduced. The quintile share mean of the poor (i.e. denominator) of the QSR is unaffected. Variance estimates are computed as first-order linearization using the design-based estimation facilities in the `survey` package.

Value

Object of class `svystat.rob`, which is scalar with a `var` attribute giving the variance, a `statistic` attribute giving the name of the statistic, a `k` attribute giving the robustness tuning constant, and a `method` attribute indicating the computation method. In addition the objects possess further attributes concerning number of observations, number of NA’s, number of declared outliers, average weight, and several details with regard to the optimization.

Author(s)

Beat Hulliger and Tobias Schoch
References


See Also

svymean, tsvymeans, msvymeans, BQSR.

Examples

```r
## Pseudo example
MQSR(eqIncome, datsilc2004, k=4)
```
Chapter 5

Multivariate Robust Imputation

The code for this chapter can be found in Part II, Chapter 9, Chapter 10 and Chapter 11.

5.1 Mahalanobis-Distance Based Robust Imputation
Description

BEM starts from a set of uncontaminated data with possible missing values, applies a version of the EM-algorithm to estimate the center and scatter of the good data, then adds (or deletes) observations to the good data which has a Mahalanobis distance below a threshold. This process iterates until the good data remain stable. Observations not among the good data are outliers.

Usage

BEM(data, weights, v = 2, c0 = 3, alpha = 0.01, md.type = "m", em.steps.start = 10, em.steps.loop = 5, better.estimation = F, steps.output = F)

Arguments

data a matrix or data frame. As usual, rows are observations and columns are variables.
weights a non-negative and non-zero vector of weights for each observation. Its length must equal the number of rows of the data. Default is rep(1,nrow(data)).
v an integer indicating the distance for the definition of the starting good subset: v=1 uses the Mahalanobis distance based on the weighted mean and covariance, v=2 uses the Euclidean distance from the component-wise median
c0 the size of initial subset is c0*ncol(data).
alpha a probability indicating the level (1-alpha) of the cutoff quantile for good observations
md.type Type of Mahalanobis distance: m marginal, c conditional
em.steps.start Number of iterations of EM-algorithm for starting good subset
em.steps.loop Number of iterations of EM-algorithm for good subset
better.estimation If better.estimation=TRUE then the EM-algorithm for the final good subset iterates em.steps.start more.
steps.output If TRUE verbose output.

© http://ameli.surveystatistics.net/ - 2011
Details

The BACON algorithm with $v=1$ is not robust but affine equivariant. The threshold for Mahalanobis distances is a chisquare quantile at $(1-\alpha)$. For relatively small data sets it may be better to choose $\alpha/n$ instead.

EM.normal is usually called from BEM. EM.normal is implementing the EM-algorithm in such a way that part of the calculations can be saved to be reused in the BEM algorithm.

Value

The output is stored in a global variable BEM$r$ with components:

- `sample.size`: number of observations
- `number.of.variables`: Number of variables
- `significance.level`: $\alpha$
- `final.basic.subset.size`: Size of final good subset
- `number.of.iterations`: Number of iterations of the BACON step
- `computation.time`: Elapsed computation time
- `good.data`: Indices of the data in the final good subset
- `outliers`: Indices of the outliers
- `center`: Final estimate of the center
- `scatter`: Final estimate of the covariance matrix
- `dist`: Final Mahalanobis distances

Note

BEM uses an adapted version of the EM-algorithm in function EM-normal.

Author(s)

Cédric Béguin and Beat Hulliger
References


Examples

```r
# Bushfire data set with 20% MCAR
data(bushfirem, bushfire.weights)
BEM(bushfirem, bushfire.weights, alpha=(1-0.01/nrow(bushfirem)))
```
GIMCD

Gaussian imputation followed by MCD

Description

Gaussian imputation uses the classical non-robust mean and covariance estimator and then imputes predictions under the multivariate normal model. Outliers may be created by this procedure. Then a high-breakdown robust estimate of the location and scatter with the Minimum Covariance Determinant algorithm is obtained and finally outliers are determined based on Mahalanobis distances based on the robust location and scatter.

Usage

GIMCD(data, alpha = 0.05, plotting = FALSE, seed = 234567819)

Arguments

data         a data frame or matrix with the data
alpha        a threshold value for the cut-off for the outlier Mahalanobis distances
plotting     if TRUE plot the Mahalanobis distances
seed         random number generator seed

Details

Normal imputation from package norm and MCD from package MASS

Value

Result is stored in a global list GIMCD.r:

center      robust center
scatter     robust covariance
dist        Mahalanobis distances
alpha       Quantile for cut-off value
outliers    Indices of outliers

Author(s)

Cédric Béguin and Beat Hulliger
References


See Also

MASS, norm

Examples

data(bushfirem)
GIMCD(bushfirem,plotting=TRUE,alpha=0.1)
POEM  

*Nearest Neighbour Imputation with Mahalanobis distance*

**Description**

POEM takes into account missing values, outlier indicators, error indicators and sampling weights.

**Usage**

POEM(data, weights, outind, errors, missing.matrix, alpha = 0.5, 
beta = 0.5, reweight.out = FALSE, c = 5, 
preliminary.mean.imputation = FALSE, verbose=FALSE)

**Arguments**

- `data`: a data frame or matrix with the data
- `weights`: sampling weights
- `outind`: an indicator vector for the outliers, 1 indicating outlier
- `errors`: matrix of indicators for items which failed edits
- `missing.matrix`: the missingness matrix can be given as input. Otherwise it will be recalculated
- `alpha`: scalar giving the weight attributed to an item that is failing
- `beta`: minimal overlap to accept a donor
- `reweight.out`: if TRUE the outliers are redefined
- `c`: tuning constant when redefining the outliers (cutoff for Mahalanobis distances)
- `preliminary.mean.imputation`: assume the problematic observation is at the mean of good observations
- `verbose`: if TRUE verbose output

**Details**

POEM assumes that an multivariate outlier detection has been carried out beforehand and assumes the result is summarized in the vector `outliers`. Preliminary mean imputation is sometimes needed to avoid a non-positive definite covariance estimate. It assumes that the problematic values of an observation (with errors, outliers or missing) can be replace by the mean of the rest of the non-problematic observations.
Value

The result is given in two global lists: \texttt{POEM.r} contains the information on POEM and \texttt{POEM.i} contains the imputed data.

Author(s)

Cédric Béguin and Beat Hulliger

References


Examples

```r
data(bushfirem)
data(bushfire.weights)
outliers<-rep(0,nrow(bushfirem))
outliers[31:38]<-1
POEM(bushfirem,bushfire.weights,outliers,prel=TRUE)
```
TRC

Transformed rank correlations for multivariate outlier detection

Description

TRC starts from bivariate Spearman correlations and obtains a positive definite covariance matrix by back-transforming robust univariate medians and mads of the eigenspace. TRC can cope with missing values by a regression imputation using the a robust regression on the best predictor and it takes sampling weights into account.

Usage

TRC(data, weight, overlap = 3, mincor = 0, robust.regression = "rank", gamma = 0.5, prob.quantile = 0.75, alpha = 0.05, md.type = "m", output = F)

Arguments

data a data frame or matrix with the data
weight sampling weights
overlap minimum number of jointly observed values for calculating the rank correlation
mincor minimal absolute correlation to impute
robust.regression type of regression: irls is iteratively reweighted least squares M-estimator, rank is based on the rank correlations
gamma minimal number of jointly observed values to impute
prob.quantile if mads are 0 try this quantile of absolute deviations
alpha (1-alpha) Quantile of F-distribution is used for cut-off
md.type Type of Mahalanobis distance when missing values occur: m marginal (default), c conditional
output if TRUE verbose output

Details

TRC is similar to a one-step OGK estimator where the starting covariances are obtained from rank correlations and an ad hoc missing value imputation plus weighting is provided.
Value

The output of TRC is stored in a global list TRC.r with components:

- **sample.size**: number of observations
- **number.of.variables**: number of variables
- **number.of.missing.items**: number of missing values
- **significance.level**: 1-alpha
- **computation.time**: elapsed computation time
- **medians**: componentwise medians
- **mads**: componentwise mads
- **center**: location estimate
- **scatter**: covariance estimate
- **robust.regression**: input parameter
- **md.type**: input parameter
- **good.data**: indices of non-outliers
- **outliers**: indices of outliers
- **dist**: Mahalanobis distances (with missing values)
- **dist.with.imputed.values**: Mahalanobis distances after ad hoc imputation
- **var.with.imputed.values**: covariance estimate with ad hoc imputations

Author(s)

Cédric Béguin and Beat Hulliger

References


Examples

```r
data(bushfirem,bushfire.weights)
TRC(bushfirem,weight=bushfire.weights)
```
plotMD

Plots Mahalanobis distances versus F-quantiles

Description

QQ-plot vs. scaled F-distribution.

Usage

plotMD(dist, p, alpha = 0.95)

Arguments

dist  a vector of Mahalanobis distances
p     the number of variables involved in the Mahalanobis distances
alpha a quantile for cut-off

Details

median(dist)*qf((1:n)/(n+1),p,n-p)/qf(0.5,p,n-p)

Value

QQ-plot

Author(s)

Beat Hulliger

References

~put references to the literature/web site here ~
weighted.var

Weighted univariate variance coping with missing values

Description

This function is as weighted.mean. The squares are weighted with \( w \) and the divisor is \( \text{sum}(w)-1 \).

Usage

weighted.var(x, w, na.rm = FALSE)

Arguments

- **x**: a vector with data
- **w**: positive weights
- **na.rm**: if TRUE remove missing values

Value

- **comp1**: Description of 'comp1'
- **comp2**: Description of 'comp2'

Author(s)

Beat Hulliger

References

~put references to the literature/web site here~

See Also

See Also as weighted.mean

Examples

```r
x <- rnorm(100)
x[sample(1:100, 20)] <- NA
w <-rchisq(100, 2)
weighted.var(x, w, na.rm = TRUE)
```
5.2 Depth Based Robust Imputation
Epidemic Algorithm for detection of multivariate outliers in incomplete survey data.

Description

In EAdet an epidemic is started at a center of the data. The epidemic spreads out and infects neighbouring points (probabilistically or deterministically). The last points infected are outliers. After running EAdet an imputation with EAimp may be run. It uses the distances calculated in EAdet and starts an epidemic at each observation to be imputed until donors for the missing values are infected. Then a donor is selected randomly.

Usage

EAdet(data, weights, reach = "max", transmission.function = "root", power = ncol(data), distance.type = "euclidean", global.distances = F, maxl = 5, plotting = T, monitor = F, prob.quantile = 0.9, random.start = F, fix.start, threshold = F, deterministic = TRUE, remove.missobs=FALSE)

EAimp(data, weights , outind=EAdet.i$outind, duration = EAdet.r$duration, maxl = 5, kdon = 1, monitor = FALSE, threshold = FALSE, deterministic = TRUE, fixedprop = 0)

Arguments

data a data frame or matrix with the data
weights a vector of positive sampling weights
reach if reach= max the maximal nearest neighbour distance is used as the basis for the transmission function, otherwise the weighted \((1-(p+1)/n)\) quantile of the nearest neighbour distances is used.

transmission.function

form of the transmission function of distance d: step is a heaviside function which jumps to 1 at d0, linear is linear between 0 and d0, power is \((beta*d+1)^(-p)\) for \(p=ncol(data)\) as default, root is the function \(1-(1-d/d0)^{-1/Maxl}\)

power sets \(p=power\)

distance.type

distance type in function dist()
global.distances
if TRUE uses the global distance stored in EA.distances instead, otherwise calculates the distances freshly

maxl  Maximum number of steps without infection

plotting if TRUE the cdf of infection times is plotted

monitor if TRUE verbose output on epidemic

prob.quantile If mads fail take this quantile absolute deviation

random.start If TRUE take a starting point at random instead of the spatial median

fix.start Force epidemic to start at a specific observation

threshold Infect all remaining points with infection probability above the threshold 1−0.5^(1/maxl)

deterministic if TRUE the number of infections is the expected number and the infected observations are the ones with largest infection probabilities.

remove.missobs Set remove.missobs to TRUE if completely missing observations should be discarded. This has to done actively as a safeguard to avoid mismatches when imputing.

duration The duration of the detection epidemic

outind a boolean vector indicating outliers

kdon The number of donors that should be infected before imputation

fixedprop If TRUE a fixed proportion of observations is infected at each step

Details

The form and parameters of the transmission function should be chosen such that the infection times have at least a range of 10. The default cutting point to decide on outliers is the median infection time plus three times the mad of infection times. A better cutpoint may be chosen by visual inspection of the cdf of infection times.

Value

EAdet with global.distances=F calls the function EA.dist, which stores the counterprobabilities of infection in a global variable EA.distances and three parameters (sample spatial median index, maximal distance to nearest neighbor and transmission distance=reach) in EA.distances.parameters. For EAdet the result is stored in two global variables: EAdet.r and EAdet.i. EAdet.r has the following components:

sample.size Number of observations

number.of.variables Number of variables
n.complete.records  
Number of records without missing values

n.usable.records  
Number of records with less than half of values missing (unusable observations are discarded)

medians  
Component wise medians

mads  
Component wise mads

prob.quantile  
Use this quantile if mads fail, i.e. if one of the mads is 0.

quantile.deviations  
Quantile of absolute deviations.

start  
Starting observation

transmission.function  
Input parameter

power  
Input parameter

min.nn.dist  maximal nearest neighbor distance

transmission.distance  
d0

threshold  
Input parameter

distance.type  
Input parameter

deterministic  
Input parameter

number.infected  
Number of infected observations

cutpoint  
Cutpoint of infection times for outlier definition

outliers  
Indices of outliers

duration  
Duration of epidemic

computation.time  
Elapsed computation time

initialisation.computation.time  
Elapsed computation time for standardisation and calculation of distance matrix

EAdet.i contains two vectors of length nrow(data):

infected  
Indicator of infection

infection.time  
Time of infection

EAimp stores the result in two global variables EAimp.r and EAimp.data. The components of EAimp.r are a subset of the components of EAdet.r and EAimp.data contains the imputed dataset.
Author(s)

Cédric Béguin and Beat Hulliger

References


Examples

data(bushfirem, bushfire.weights)
EAdet(bushfirem, bushfire.weights)
EAimp(bushfirem, mon=TRUE, kdon=3)
**Description**

The ER function is an implementation of the ER-algorithm of Little and Smith (1987).

**Usage**

```r
ER(data, weights, alpha = 0.01, psi.par = c(2, 1.25), em.steps = 100, steps.output = F, Estep.output=F)
```

**Arguments**

- `data` a data frame or matrix
- `weights` sampling weights
- `alpha` probability for the quantile of the cut-off
- `psi.par` further parameters passed to the psi-function
- `em.steps` number of iteration steps of the EM-algorithm
- `steps.output` if TRUE verbose output
- `Estep.output` if TRUE estimators are output at each iteration

**Details**

The M-step of the EM-algorithm uses a one-step M-estimator.

**Value**

The output is stored in a global variable `ER.r` with components:

- `sample.size` number of observations
- `number.of.variables` Number of variables
- `significance.level` alpha
- `computation.time` Elapsed computation time

© http://ameli.surveystatistics.net/ - 2011
good.data  Indices of the data in the final good subset
outliers   Indices of the outliers
center    Final estimate of the center
scatter   Final estimate of the covariance matrix
dist      Final Mahalanobis distances
robweights Robustness weights in the final EM step

Author(s)

Beat Hulliger

References


See Also

BEM

Examples

data(bushfirem)
data(bushfire.weights)
ER(bushfirem, weights=bushfire.weights, alpha=0.01, steps.output=TRUE)
5.3 Robust Imputation for Compositions
Description

Computes the Aitchison distance between two observations or between two data sets.

Usage

aDist(x, y)

Arguments

x a vector, matrix or data.frame
y a vector, matrix or data.frame with equal dimension as x

Details

This distance measure accounts for the relative scale property of the Aitchison distance. It measures the distance between two compositions if x and y are vectors and evaluate sum of the distances between x and y for each row of x and y if x and y are matrices or data frames.

It is not designed to apply it on one matrix, such as function ‘acomp()’ in package ‘compositions’, but it is designed to compare different matrices.

The underlying code is written in C and allows a fast computation also for large data sets.

Value

The Aitchison distance between two compositions or between two data sets.

Author(s)

Matthias Templ
References


See Also

*ilr*

Examples

```r
data(expenditures)
x <- xOrig <- expenditures
## Aitchison distance between the first 2 observations:
aDist(x[,1], x[,2])

## set some missing values:

## impute them:
xImp <- impCoda(x, method="ltsReg")$xImp

## calculate the relative Aitchison distance between xOrig and xImp:
aDist(xOrig, xImp)
```
Description

This function offers different methods for the imputation of missing values in compositional data. Missing values are initialized with proper values. Then iterative algorithms try to find better estimations for the former missing values.

Usage

```r
impCoda(x, maxit = 10, eps = 0.5, method = "ltsReg", closed = FALSE, init = "KNN", k = 5, dl = rep(0.05, ncol(x)), noise=0.1)
```

Arguments

- `x`: data frame or matrix
- `maxit`: maximum number of iterations
- `eps`: convergence criteria
- `method`: imputation method
- `closed`: imputation of transformed data (using ilr transformation) or in the original space (closed equals TRUE)
- `init`: method for initializing missing values
- `k`: number of nearest neighbors (if init $==$ "KNN")
- `dl`: detection limit(s), only important for the imputation of rounded zeros
- `noise`: amount of adding random noise to predictors after convergency

Details

eps: The algorithm is finished as soon as the imputed values stabilize, i.e. until the sum of Aitchison distances from the present and previous iteration changes only marginally (eps).

method: Several different methods can be chosen, such as `ltsReg`: least trimmed squares regression is used within the iterative procedure. `lm`: least squares regression is used within the iterative procedure. `classical`: principal component analysis is used within the iterative procedure. `ltsReg2`: least trimmed squares regression is used within the iterative procedure. The imputed values are perturbed in the direction of the predictor by values drawn from a normal distribution with mean and standard deviation related to the corresponding residuals and multiplied by `noise`.

method `roundedZero` is experimental. It imputes rounded zeros within our iterative framework.
Value

xOrig Original data frame or matrix
xImp Imputed data
criteria Sum of the Aitchison distances from the present and previous iteration
iter Number of iterations
maxit Maximum number of iterations
w Amount of imputed values
wind Index of the missing values in the data

Author(s)

Matthias Templ, Karel Hron

References


See Also

impKNNa, ilr

Examples

data(expenditures)
x <- expenditures
x[1,3]
x[1,3] <- NA
xi <- impCoda(x)$xImp
xi[1,3]
s1 <- sum(x[1,-3])
impS <- sum(xi[1,-3])
xi[,3] * s1/impS
impKNNa

Imputation of missing values in compositional data using knn methods

Description

This function offers several k-nearest neighbor methods for the imputation of missing values in compositional data.

Usage

```r
impKNNa(x, method = "knn", k = 3, metric = "Aitchison", agg = "median",
primitive = FALSE, normknn = TRUE, das = FALSE, adj="median")
```

Arguments

- `x`: data frame or matrix
- `method`: method (at the moment, only `knn` can be used)
- `k`: number of nearest neighbors chosen for imputation
- `metric`: Aitchison or Euclidean
- `agg`: median or mean, for the aggregation of the nearest neighbors
- `primitive`: if TRUE, a more enhanced search for the $k$-nearest neighbors is obtained (see details)
- `normknn`: An adjustment of the imputed values is performed if TRUE
- `das`: depreciated. if TRUE, the definition of the Aitchison distance, based on simple logratios of the compositional part, is used (Aitchison, 2000) to calculate distances between observations. if FALSE, a version using the clr transformation is used.
- `adj`: either median (default) or sum can be chosen for the adjustment of the nearest neighbors, see Hron et al., 2010.

Details

The Aitchison metric should be chosen when dealing with compositional data, the Euclidean metric otherwise.

If primitive == FALSE, a sequential search for the $k$-nearest neighbors is applied for every missing value where all information corresponding to the non-missing cells plus the information in the variable to be imputed plus some additional information is available. If primitive == TRUE, a search of the $k$-nearest neighbors among
impKNNa

observations is applied where in addition to the variable to be imputed any further
cells are non-missing.

If normknn is TRUE (prefered option) the imputed cells from a nearest neighbor
method are adjusted with special adjustment factors (more details can be found online
(see the references)).

Value

<table>
<thead>
<tr>
<th>xOrig</th>
<th>Original data frame or matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>xImp</td>
<td>Imputed data</td>
</tr>
<tr>
<td>w</td>
<td>Amount of imputed values</td>
</tr>
<tr>
<td>wind</td>
<td>Index of the missing values in the data</td>
</tr>
<tr>
<td>metric</td>
<td>Metric used</td>
</tr>
</tbody>
</table>

Author(s)

Matthias Templ

References

Aitchison, J. and Barcelo-Vidal, C. and Martin-Fernandez, J.A. and Pawlowsky-

positional data using classical and robust methods Computational Statistics and Data
Analysis, In Press, Corrected Proof, ISSN: 0167-9473, DOI:10.1016/j.csda.2009.11.023

See Also

impCoda

Examples

data(expenditures)
x <- expenditures
x[1,3] <- NA
xi <- impKNNa(x)$xImp
xi[1,3]
**invilr**  
*Inverse isometric log-ratio transformation*

**Description**

The inverse transformation of ilr().

**Usage**

`invilr(x.ilr)`

**Arguments**

- `x.ilr`: data frame or matrix

**Details**

For details on the choice of the balances, please, see at the research report for which the link is given below.

**Value**

The transformed data.

**Author(s)**

Karel Hron

**References**


**See Also**

`ilr`
Examples

```r
require(MASS)
Sigma <- matrix(c(5.05, 4.95, 4.95, 5.05), ncol=2, byrow=TRUE)
set.seed(123)
z <- mvrnorm(100, mu=c(0,2), Sigma=Sigma)
invilr(z)
```
**ilr**  
*Isometric log-ratio transformation*

**Description**

An isometric log-ratio transformation with a special choice of the balances according to Hron et al. (2010).

**Usage**

```r
ilr(x)
```

**Arguments**

- `x` object of class data.frame or matrix with positive entries

**Details**

The ilr transformation moves D-part compositional data from the simplex into a (D-1)-dimensional real space isometrically. From this choice of the balances, all the relative information of the part $x_1$ from the remaining parts is separated. It is useful for estimating missing values in $x_1$ by regression of the remaining variables.

**Value**

The ilr transformed data.

**Author(s)**

Karel Hron, Matthias Templ

**References**


See Also

-invilr, ilr

Examples

require(MASS)
Sigma <- matrix(c(5.05, 4.95, 4.95, 5.05), ncol=2, byrow=TRUE)
z <- invilr(mvrnorm(100, mu=c(0,2), Sigma=Sigma))
Description

Estimates the variation matrix with robust methods.

Usage

robVariation(x, robust=TRUE)

Arguments

x data frame or matrix with positive entries
robust if FALSE, standard measures are used.

Details

The variation matrix is estimated for a given compositional data set. Instead of using the classical standard deviations the mad is used when parameter robust is set to TRUE.

Value

The (robust) variation matrix.

Author(s)

Matthias Templ

References


See Also

variation
Examples

data(expenditures)
robVariation(expenditures)
robVariation(expenditures, robust=FALSE)
5.4 Sequential Robust Imputation
Iterative robust model-based imputation (IRMI)

Description

In each step of the iteration, one variable is used as a response variable and the remaining variables serve as the regressors.

Usage

```r
irmi(x, eps = 0.01, maxit = 100, mixed = NULL, step = FALSE,
    robust = FALSE, takeAll = TRUE, noise = TRUE, noise.factor = 1,
    force = FALSE, robMethod = "lmrob", force.mixed = TRUE, mi = 1,
    trace=FALSE)
```

Arguments

- **x**: data.frame or matrix
- **eps**: threshold for convergency
- **maxit**: maximum number of iterations
- **mixed**: column index of the semi-continuous variables
- **step**: a stepwise model selection is applied when the parameter is set to TRUE
- **robust**: if TRUE, robust regression methods will be applied
- **takeAll**: takes information of (initialised) missings in the response as well for regression imputation.
- **noise**: irmi has the option to add a random error term to the imputed values, this creates the possibility for multiple imputation. The error term has mean 0 and variance corresponding to the variance of the regression residuals.
- **noise.factor**: amount of noise.
- **force**: if TRUE, the algorithm tries to find a solution in any case, possible by using different robust methods automatically.
- **robMethod**: regression method when the response is continuous.
- **force.mixed**: if TRUE, the algorithm tries to find a solution in any case, possible by using different robust methods automatically.
- **mi**: number of multiple imputations.
- **trace**: Additional information about the iterations when trace equals TRUE.
Details

The method works sequentially and iterative. The method can deal with a mixture of continuous, semi-continuous, ordinal and nominal variables including outliers.

A full description of the method will be uploaded soon in form of a package vignette.

Value

the imputed data set.

Author(s)

Matthias Templ, Alexander Kowarik

See Also

mi

Examples

data(sleep)
irmi(sleep)
Bibliography


Part II

Code
msvymean.survey.design <- function(y, design, type=c("rht", "rwm"), na.rm=T, k=3, steps=50, plot=F, acc=1e-6, quietly=F, psi=huberPsi, exact=F) {
call <- match.call()
psi.function <- ifelse(is.null(call$psi), "huberPsi", paste(call$psi))
type <- match.arg(type)
method <- switch(type,
  rht = "robust Horvitz-Thompson estimator",
  rwm = "robust weighted mean")
add.method <- ifelse(steps==1, " (One-step M-estimation)", " (M-estimation)"
method <- paste(method, add.method, sep="")
original.design <- design
# check for survey.design and na
check <- .rsvycheck(y, design, na.rm=na.rm)
design <- check$design
# allow only for one study variable
if (dim(check$mat)[2] > 2) stop("only one 'y'-variable allowed")
x <- switch(type,
  rht = mean(check$mat[,2]) / check$mat[,2],
  rwm = rep(1, NROW(check$mat)))
# mat: matrix[, c(y, weights, x)]
mat <- cbind(check$mat, as.matrix(x))
nas <- check$na
method <- paste(method, check$method, sep="")
# compute the quantiles
x.quant <- .svyquantiles(x=mat[,3], w=mat[,2], probs=c(0.5, 0.8))
y.quant <- .svyquantiles(x=mat[,1], w=mat[,2], probs=c(0.5, 0.8))
mad.y <- .svyquantiles(abs(mat[,1] - y.quant[1]), mat[,2], probs=0.5)*1.4826
# set the starting values
if (x.quant[1] == 0) {
  beta.med <- y.quant[1]
  initial <- round(y.quant[1], 3)
} else {
  initial <- round(beta.med, 3)
}
if (beta.med == 0) {
}
# call irls
Chapter 6. RHT Package

```r
result <- .irls(k, mat, acc, steps, design, beta.med, mer=FALSE, true.mean=NULL, psi=psi)
if(plot){
  sel <- result$u<1
  if(length(which(sel == TRUE))> 2){
    op <- par(mfcol=c(2,2), mar=c(2,4,4,0))
    par(mar=c(2,4,4,0))
    xx <- mat[,3]
    top <- max(result$ures[sel]/sqrt(xx[sel]))
    bottom <- min(result$ures[sel]/sqrt(xx[sel]))
    oldscale <- (mat[,1] - beta.med * mat[,3]) / sqrt(mat[,3])
    plotrange <- range(oldscale, result$ures/sqrt(mat[,3]))
    plot(oldscale ~ rep(1.2, length(oldscale)), main="Standardized residuals",
    ylab="stand. res.", xlab="x", col="blue", ylim=plotrange, xlin=c(1,2), axes=F)
    axis(2)
    axis(1, at=c(1.2, 1.8), labels=c("initial", "robust"))
    box()
    points(result$ures/sqrt(mat[,3]) ~ rep(1.8, length(oldscale)), col="red")
    abline(h=top, lty=2)
    abline(h=bottom, lty=2)
    # plot robustness weights
    par(mar=c(2,4,4,0))
    plot(mat[,1]/mat[,3], result$u, main="Robustness weights", ylab="rob. weights",
    xlab=paste(colnames(mat)[1]), type="p", col="blue")
    # density plot / std.res vs. x plot
    if(type=="rwm"){
      par(mar=c(2,4,4,10))
      plot(density(result$ures/sqrt(mat[,3]))$y, density(result$ures/sqrt(mat[,3]))$x,
      ylim=plotrange, type="l", axes=F, col="red", xlab="", ylab="",
      main="Density of the std. res.")
      lines(density(oldscale)$y, density(oldscale)$x, ylim=plotrange, col="blue")
    } else{
      par(mar=c(2,4,4,1))
      svyplot(oldscale ~ mat[,3], design=design, main="Std. res. vs. u[i] / sum(u[i])",
      ylab="stand. res.", xlab="x", fg="blue", yim=plotrange)
      points(result$ures/sqrt(mat[,3]) ~ mat[,3], col="red", pch=22)
      abline(h=top, lty=2)
      abline(h=bottom, lty=2)
    }
    # "plot" specification
    par(mar=c(4,4,4,0))
    plot(c(0,10), ylim=c(0,10), ylab=c(0,10), axes=F, pch="", xlab="", ylab="")
    text(0,10, "Method:", pos=4, font=2)
    text(0.9, paste(method), pos=4)
    text(0.8, "Robustness parameter (k):", pos=4, font=2)
    text(0.7, paste(k), pos=4)
    text(0.6, "Number of outliers:", pos=4, font=2)
    text(0.5, paste(round(sum(result$u<1))), pos=4)
    text(0.3, "Legend:", pos=4, font=2)
    legend(0.3, col=c("blue", "red"), legend=c("initial", "robust"), pch=c(1,22), ncol=1, bg="white", bty="n")
  }
  par(op)
} else{
  par(mfcol=c(1,2), mar=c(3,4,3,4))
  svyplot(result$ures/sqrt(mat[,3]) ~ mat[,3], design=design, main="Standardized
  residual vs. 'x'\nnew scale",
  ylab="stand. res.", xlab="x")
}
par(op)
}
else{
  average <- result$d
  names(average) <- colnames(mat)[1]
  # IF exact=TRUE: the variance estimates considers the
  # MAD as initial scale estimate, ELSE: variance estimates
  # comes from irls
  if(exact){
    result <- .irls(k, mat, acc, steps, design, beta.med, mer=FALSE, true.mean=NULL, psi=psi)
    sel <- result$u<1
    if(length(which(sel == TRUE))> 2){
      op <- par(mfcol=c(2,2), mar=c(2,4,4,0))
      par(mar=c(2,4,4,0))
      xx <- mat[,3]
      top <- max(result$ures[sel]/sqrt(xx[sel]))
      bottom <- min(result$ures[sel]/sqrt(xx[sel]))
      oldscale <- (mat[,1] - beta.med * mat[,3]) / sqrt(mat[,3])
      plotrange <- range(oldscale, result$ures/sqrt(mat[,3]))
      plot(oldscale ~ rep(1.2, length(oldscale)), main="Standardized residuals",
      ylab="stand. res.", xlab="x", col="blue", ylim=plotrange, xlin=c(1,2), axes=F)
      axis(2)
      axis(1, at=c(1.2, 1.8), labels=c("initial", "robust"))
      box()
      points(result$ures/sqrt(mat[,3]) ~ rep(1.8, length(oldscale)), col="red")
      abline(h=top, lty=2)
      abline(h=bottom, lty=2)
      # plot robustness weights
      par(mar=c(2,4,4,0))
      plot(mat[,1]/mat[,3], result$u, main="Robustness weights", ylab="rob. weights",
      xlab=paste(colnames(mat)[1]), type="p", col="blue")
      # density plot / std.res vs. x plot
      if(type=="rwm"){
        par(mar=c(2,4,4,10))
        plot(density(result$ures/sqrt(mat[,3]))$y, density(result$ures/sqrt(mat[,3]))$x,
        ylim=plotrange, type="l", axes=F, col="red", xlab="", ylab="",
        main="Density of the std. res.")
        lines(density(oldscale)$y, density(oldscale)$x, ylim=plotrange, col="blue")
      } else{
        par(mar=c(2,4,4,1))
        svyplot(oldscale ~ mat[,3], design=design, main="Std. res. vs. u[i] / sum(u[i])",
        ylab="stand. res.", xlab="x", fg="blue", yim=plotrange)
        points(result$ures/sqrt(mat[,3]) ~ mat[,3], col="red", pch=22)
        abline(h=top, lty=2)
        abline(h=bottom, lty=2)
      }
      # "plot" specification
      par(mar=c(4,4,4,0))
      plot(c(0,10), ylim=c(0,10), ylab=c(0,10), axes=F, pch="", xlab="", ylab="")
      text(0,10, "Method:", pos=4, font=2)
      text(0.9, paste(method), pos=4)
      text(0.8, "Robustness parameter (k):", pos=4, font=2)
      text(0.7, paste(k), pos=4)
      text(0.6, "Number of outliers:", pos=4, font=2)
      text(0.5, paste(round(sum(result$u<1))), pos=4)
      text(0.3, "Legend:", pos=4, font=2)
      legend(0.3, col=c("blue", "red"), legend=c("initial", "robust"), pch=c(1,22), ncol=1, bg="white", bty="n")
    }
    par(op)
  }
  else{
    average <- result$d
    names(average) <- colnames(mat)[1]
    # IF exact=TRUE: the variance estimates considers the
    # MAD as initial scale estimate, ELSE: variance estimates
    # comes from irls
    if(exact){
      result <- .irls(k, mat, acc, steps, design, beta.med, mer=FALSE, true.mean=NULL, psi=psi)
      sel <- result$u<1
      if(length(which(sel == TRUE))> 2){
        op <- par(mfcol=c(2,2), mar=c(2,4,4,0))
        par(mar=c(2,4,4,0))
        xx <- mat[,3]
        top <- max(result$ures[sel]/sqrt(xx[sel]))
        bottom <- min(result$ures[sel]/sqrt(xx[sel]))
        oldscale <- (mat[,1] - beta.med * mat[,3]) / sqrt(mat[,3])
        plotrange <- range(oldscale, result$ures/sqrt(mat[,3]))
        plot(oldscale ~ rep(1.2, length(oldscale)), main="Standardized residuals",
        ylab="stand. res.", xlab="x", col="blue", ylim=plotrange, xlin=c(1,2), axes=F)
        axis(2)
        axis(1, at=c(1.2, 1.8), labels=c("initial", "robust"))
        box()
        points(result$ures/sqrt(mat[,3]) ~ rep(1.8, length(oldscale)), col="red")
        abline(h=top, lty=2)
        abline(h=bottom, lty=2)
      }
      else{
        par(mfcol=c(1,2), mar=c(3,4,3,4))
        svyplot(result$ures/sqrt(mat[,3]) ~ mat[,3], design=design, main="Standardized
        residual vs. 'x'\nnew scale",
        ylab="stand. res.", xlab="x")
      }
    }
  }
}
```
linearized.values <- .inflmest(x=mat[,1], weight=mat[,2], T=result$m, k=k, S=mad.y, psi=psi)
d <- update(design, zz=linearized.values)
attr(average, "var") <- vcov(svymean(~zz, d))
}

```
for (i in seq_len(NROW(mat))) {
  x <- switch(type, 
    rht = mean(check$mat[,2]) / check$mat[,2], 
    rwm = rep(1, NROW(check$mat)))
  mat <- cbind(check$mat, as.matrix(x))
  nas <- check$na
  x.quant <- .svyquantiles(x=mat[,3], w=mat[,2], probs=c(0.5, 0.8))
  y.quant <- .svyquantiles(x=mat[,1], w=mat[,2], probs=c(0.5, 0.8))
  if (x.quant[1] == 0){
    beta.med <- y.quant[1]
    initial <- round(y.quant[1], 3)
  } else{
    initial <- round(beta.med, 3)
  }
  if (beta.med == 0){
  }
  true.mean <- coef(svymean(eval(y.variable), design, na.rm=TRUE, quietly=TRUE))
```
Chapter 6. RHT Package

```r
opt <- optim(par=init, .irls, mat=mat, acc=acc, steps=steps, design=design, 
beta.med=beta.med, mer=TRUE, true.mean=true.mean, psi=psi, method="L-BFGS-B", 
lower=box.lo, control=list(REPORT=2, trace=1))
if(opt$convergence==0 | opt$ convergence==51){
  mer.estimate <- .irls(k=opt$par, mat=mat, acc=acc, steps=steps, design=design, 
beta.med=beta.med, mer=FALSE, true.mean=NULL, psi=psi)
average <- mer.estimate$m
names(average) <- colnames(mat)[1]
attr(average, "nobs") <- NROW(mat)
attr(average, "na") <- nas
attr(average, "method") <- paste("Minimum estimated risk estimation, based on \m", 
                     method, sep="")
attr(average, " statistic ") <- " mean 
attr(average, " outliers ") <- sum(mer.estimate$u<1)
attr(average, " design ") <- design
attr(average, " call ") <- match.call()
attr(average, " type ") <- type
rob <- cbind(attr(object, " rob ")[[1]], round(mean(mer.estimate$u),5), round(opt$par,5))
colnames(rob) <- c(" psi ", " ave. weight ", " optimal k")
rownames(rob) <- " ...
attr(average, " rob ") <- rob
optim <- cbind(opt$counts[[1]], round(opt$value, 1), acc, init, box.lo)
colnames(optim) <- c("MER - iter ", "mse", "precision", "initial", "low box.constr.")
rownames(optim) <- "L-BFGS-B"
attr(average, " optim ") <- optim
class(average) <- c("svystat.rob", " mer ")
summary(average)
return(average)
} else{
  cat("no output generated because not converged! \n")
  if(opt$convergence==1) cat("maximum iterations reached, reduce tolerance < 1e-4 \n")
  if(opt$ convergence > 5) cat("L-BFGS-B warning: ", opt$message, "\n")
  cat("see help for modifications of the initial estimate (init), \nthe lower box constraint (box.lo), and the numerical convergence \n tolerance (tol)\n ")
}
```

# FIXME: asymmetric -> psi

---

# Subject:  S3 plot function for mer objects
# Author:  Tobias Schoch
# Status:  v1.1 (mature)
# Date:  February 8 2009
# Dependency:  survey

---

# INPUT
# object  mer class
# n  number of evaluation points
# scale  scale of draw/evaluation region around optimum

```r
plot.mer <- function(object, n=10, scale=1){
  mean <- numeric(n)
  variance <- numeric(n)
  y.variable <- names(object)
  design <- attr(object, " design ")
  optimal.k <- as.numeric(attr(object, " rob ")[[3]])
  acc <- attr(object, " optim ")[1,3]
  type <- attr(object, " type ")
  method <- attr(object, " method ")
  psi <- attr(m, " rob ")[[1]]
  asymmetric <- ifelse(psi="’asym. Huber‘", TRUE, FALSE)
  true.mean <- coef(svymean(as.formula(paste("-", y.variable))), design, na.rm=TRUE, 
                     quietly=TRUE))
  lo <- max(optimal.k - scale*optimal.k)^(-1/2), 0.00001)
  hi <- min(optimal.k + scale*optimal.k)^(-1/2), 50)
  at <- seq.int(lo, hi, length.out=n)
  for(i in 1:n){
    current <- msvymean(as.formula(paste("-", y.variable)), design=design, 
                     k=at[i],
```
type = type, na.rm = TRUE, acc = acc, plot = FALSE, quietly = TRUE, asymmetric = asymmetric

mean[i] <- coef(current)
variance[i] <- vcov(current)

status[i, n]

mse <- variance + (mean - true.mean)^2

# plot mse
op <- par(mfcol = c(2,2), mar = c(4,4,4,4))
plot(at, mse, xlab="robustness parameter 'k'", ylab="estimated risk (mse)", type="l")
abline(v=optimal.k, col="red")

# plot mean
plot(at, mean, xlab="robustness parameter 'k'", ylab="mean", type="l")
abline(v=optimal.k, col="red")

# plot variance
plot(at, variance, xlab="robustness parameter 'k'", ylab="variance", type="l")
abline(v=optimal.k, col="red")

# plot legend
par(mar = c(4,0,4,0))
plot(c(0,10), xlim=c(0,10), ylim=c(0,10), axes=F, pch="", xlab="", ylab="")
text(0,10, " Method :", pos=4, font=2)
text(0,8, paste(method), pos=4)
par(op)

# Subject: Ratio M-estimation
# Author: Tobias Schoch
# Status: v1.1
# Date: March 2009
# Depends: survey
# Description: see Rd-file

msvyratio.survey.design <- function(numerator, denominator, design, na.rm=T, k=3, steps=50, plot=F, acc=1e-6, quietly=FALSE){
require(survey)

if(!is(design, "survey.design2")){
stop("Design must be an object of the 'survey.design' class\n")}
original.design <- design
method <- "robust ratio estimator"
add.method <- ifelse(steps==1, " (One-step M-estimation)", " (M-estimation)"

if(design$call[[1]] == "subset"{
method <- paste(method, add.method, sep="")
}

if(inherits(numerator, "formula")){
num <- model.frame(numerator, design$variables, na.action=na.pass)
if(NCOL(num) > 1){
stop("Only one 'y'-variable allowed!\n")
}
mat <- matrix(nrow=NROW(num), ncol=3)
mat[,1] <- as.matrix(num)
mat[,2] <- as.matrix(weights(design))

else{
stop("Numerator must be a formula object!\n")
}

if(inherits(denominator, "formula")){
den <- model.frame(denominator, design$variables, na.action=na.pass)
if(NCOL(den) > 1){
stop("Only one 'y'-variable allowed!\n")
}
mat[,3] <- as.matrix(den)
colnames(mat) <- c("numerator", "w", "denominator")
}
else{
stop("Denominator must be a formula object!\n")
}

if(na.rm){

nas <- rowSums(is.na(mat))

© http://ameli.surveystatistics.net/ - 2011
design <- design[nas == 0]
mat <- mat[nas == 0, , drop=F]
na <- length(which(nas!=0))
}
if(any(mat[,2] == 0)) cat("There are weights=0\n")
if(any(mat[,3] == 0)) cat("There are denominators=0\n")
# compute the quantiles
x.quant <- .svyquantiles(x=mat[,3], w=mat[,2], probs=c(0.5, 0.8))
y.quant <- .svyquantiles(x=mat[,1], w=mat[,2], probs=c(0.5, 0.8))
# set the starting values
if (x.quant[1] == 0){
  beta.med <- y.quant[1]
  initial <- round(y.quant[1], 3)
}
else{
  initial <- round(beta.med, 3)
}
if (beta.med == 0){
}
psi <- "Huber"
result <- .irls(k, mat, acc, steps, design, beta.med, mer=FALSE, true.mean=NULL)
if(plot){
  sel <- result$ure<1
  if(length(which(sel == TRUE)) > 2){
    op <- par(mfcol=c(2,2), mar=c(2,4,4,0))
    par(mar=c(2,4,4,0))
    xx <- mat[,3]
    top <- max(result$ures[sel]/sqrt(xx[sel]))
    bottom <- min(result$ures[sel]/sqrt(xx[sel]))
    plotrange <- range(result$ures/sqrt(mat[,3]))
    plot(oldscale = rep(1.2, length(oldscale)), main="Standardized residuals",
    ylab="stand. res.", xlab="x", col="blue", ylim=plotrange, xlim=c(1,2), axes=F)
    axis(2)
    axis(1, at=c(1.2, 1.8), labels=c("initial", "robust"))
    box()
    points(result$ures/sqrt(mat[,3]) ~ rep(1.8, length(oldscale)), col="red")
    abline(h=top, lty=2)
    abline(h=bottom, lty=2)
    # plot robustness weights
    par(mar=c(4,4,4,0))
    plot(mat[,1]/mat[,3], result$ure, main="Robustness weights", ylab="rob. weights",
    xlab="ratio", type="p", col="blue")
    # std.res vs. x plot
    par(mar=c(2,4,4,1))
    svyplot(oldscale ~ mat[,3], design=design, main="Std. res. vs. 'denominator'",
    ylab="stand. res.", xlab="x", fg="blue", ylim=plotrange)
    points(result$ures/sqrt(mat[,3]) ~ mat[,3], col="red", pch=22)
    abline(h=0, lty=2)
    abline(h=bottom, lty=2)
    # "plot" specification
    par(mar=c(4,4,4,0))
    plot(c(0,10), ylim=c(0,10), ylab="", axes=F, pch="", xlab="", ylab="")
    text(0.10, "Method:", pos=4, font=2)
    text(0.9, "Robustness parameter (k):", pos=4, font=2)
    text(0.7, paste(k), pos=4)
    text(0.6, "Number of outliers:", pos=4, font=2)
    text(0.5, paste(round(sum(result$ure<1))), pos=4)
    text(0.3, "Legend:", pos=4, font=2)
    legend(0.3, col=c("blue", "red"), legend=c("initial", "robust"), pch=c(1,22),
    ncol=1, bg="white", bty="n")
    par(op)
  }
  else{
    op <- par(mfcol=c(1,2), mar=c(3,4,3,4))
    svyplot(result$ures/sqrt(mat[,3]) ~ mat[,3], design=design,
    main="Standardized residual vs. 'x\n1 new scale'", ylab="stand. res.", xlab="x")
    par(op)
  }
}
function(y, design, trim=c(0, 1), type=c("trim", "win"), 
na.rm=T, quietly=FALSE)
if((trim[1] > trim[2]) stop("Trimming proportions not correct \n")
if((trim[1] < 0) stop("Trimming proportions not correct \n")
if((trim[2] > 1) stop("Trimming proportions not correct \n")
type <- match.arg(type)
method <- switch(type, 
trim = "trimmed weighted mean",
win = "winsorized weighted mean")
check <- .rsvycheck(y, design, na.rm=na.rm)
if(dim(check$m)[2] > 2) stop("Only one single 'y'-variable \n")
method <- paste(method, check$m, sep="")
x <- check$mat[,1:(dim(check$m)[2] - 1)]
w <- as.vector(check$m[, dim(check$m)[2] - 1])
if(type="trim"){
  average <- .trimmedmean(x, w, trim[1], trim[2])
v <- svyrecvar(.infltrim(x, w, trim[1], trim[2]))/sum(w), design$cluster, 
design$strata, design$fpc, postStrata = design$postStrata)
}
if(type="win"){
  average <- .winsorizedmean(x, w, trim[1], trim[2])
v <- svyrecvar(.inflwin(x, w, trim[1], trim[2]))/sum(w), design$cluster, 
design$strata, design$fpc, postStrata = design$postStrata)
}
```r
} else{
    rob <- cbind(trim[1], trim[2])
    colnames(rob) <- c("lo", "hi")
}
rob.names <- switch(type,
    trim = "trimming",
    win = "winsorized")
rownames(rob) <- rob.names
attr(average, "rob") <- rob
class(average) <- "svystat.rob"
if(!quietly) summary(average)
return(average)
```

```r
# Subject: S3 summary method for "svystat.rob" objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# ----------------------------------------------------------------
# INPUT
# object object of class svystat.rob
# ----------------------------------------------------------------
summary.svystat.rob <- function(object){
    if(length(object)==1) stop("no summary available
")
cat("\n")
```

```r
# Subject: S3 summary method for "svystat.rob" objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
# ----------------------------------------------------------------
# INPUT
# object object of class svystat.rob
# ----------------------------------------------------------------
summary.svystat.rob <- function(object){
    if(length(object)==1) stop("no summary available
")
cat("\n")
```

```r
# Subject: S3 summary method for "svystat.rob" objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
# ----------------------------------------------------------------
# INPUT
# object object of class svystat.rob
# ----------------------------------------------------------------
summary.svystat.rob <- function(object){
    if(length(object)==1) stop("no summary available
")
cat("\n")
```

```r
# Subject: S3 summary method for "svystat.rob" objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
# ----------------------------------------------------------------
# INPUT
# object object of class svystat.rob
# ----------------------------------------------------------------
summary.svystat.rob <- function(object){
    if(length(object)==1) stop("no summary available
")
cat("\n")
```

```r
# Subject: S3 summary method for "svystat.rob" objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
# ----------------------------------------------------------------
# INPUT
# object object of class svystat.rob
# ----------------------------------------------------------------
summary.svystat.rob <- function(object){
    if(length(object)==1) stop("no summary available
")
cat("\n")
```

```r
# Subject: S3 summary method for "svystat.rob" objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
# ----------------------------------------------------------------
# INPUT
# object object of class svystat.rob
# ----------------------------------------------------------------
summary.svystat.rob <- function(object){
    if(length(object)==1) stop("no summary available
")
cat("\n")
```
```r
# Subject: S3 print method for svystat.rob objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
#
# INPUT
# object object of class svystat.rob
#---------------------------------------------------------------
print.svystat.rob <- function(object){
m <- cbind(object, sqrt(attr(object, "var")), attr(object, "outliers"), attr(object, "nobs"), attr(object, "na"))
colnames(m) <- c(attr(object, "statistic"), "SE", "outliers", "nobs", "NA's")
printCoefmat(m, digits=7, zero.print=T)
}
```

```r
# Subject: S3 plot method for svystat.rob objects
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
#
# INPUT
# object object of svystat.rob class
#---------------------------------------------------------------
plot.plotable <- function(object){
  if(!inherits(object, "plotable")) stop("no plot object \n")
c <- attr(object, "call")
if(c[[1]]=="msvyratio.survey.design"){
  acc <- attr(object, "optim")[3]
  design <- attr(object, "design")
  k <- as.numeric(attr(object, "rob")[3])
  get.narm <- c$narm
  narm <- ifelse(is.null(get.narm), "T", paste(get.narm))
  get.steps <- c$steps
  steps <- ifelse(is.null(get.steps), 50, get.steps)
  msvyratio(eval(c$numerator), eval(c$denominator), design=design, na.rm=narm, k=k, steps=steps, plot=T, acc=acc)
}
if(c[[1]]=="msvymean.survey.design"){
  y <- attr(object, "names")
  acc <- attr(object, "optim")[3]
  design <- attr(object, "design")
  k <- as.numeric(attr(object, "rob")[3])
  gettype <- c$type
  type <- ifelse(is.null(gettype), "rht", gettype)
  get.narm <- c$narm
  narm <- ifelse(is.null(get.narm), "T", paste(get.narm))
  get.steps <- c$steps
```

© http://ameli.surveystatistics.net/ - 2011
```r
steps <- ifelse(is.null(get.steps), 50, get.steps)
huber <- attr(object, "rob")[1]
if(huber == "asym. Huber"){
    msvymean(eval(c$y), design=design, type=type, na.rm=narm, k=k, steps=steps, plot=T, acc=acc, asymmetric=TRUE)
} else{
    msvymean(eval(c$y), design=design, type=type, na.rm=narm, k=k, steps=steps, plot=T, acc=acc, asymmetric=FALSE)
}
}
```

---

```r
#' trimmedmean <- function(x, weight, lo=0, hi=1){
# xilo <- .svyquantiles(x, weight, probs=lo)
# xihi <- .svyquantiles(x, weight, probs=hi)
# res <- sum(weight[x>=xilo & x<= xihi]) / sum(weight)
# return(res)

#' winsorizedmean <- function(x, weight, lo=0, hi=1){
# xilo <- .svyquantiles(x, weight, probs=lo)
# xihi <- .svyquantiles(x, weight, probs=hi)
# x[x < xilo] <- xilo
# x[x > xihi] <- xihi
# res <- sum(weight*x) / sum(weight)
# return(res)

#' svyquantiles <- function(x, w, ties=NULL, probs = 0.5, method = "constant", f = 0)
```

---

```r
# Subject: Trimmable mean (inverse probability weighted)
# Author: Tobias Schoch
# Status: Vi.1.1 (mature)
# Date: February 8 2009
# Dependency: survey

```
ord <- order(x)
x <- x[ord]
w <- w[ord]
if(is.null(ties)){
  w.ord <- cumsum(w) / sum(w)
} else{
  w <- rowsum(w, ties, reorder = F)
x <- sort(aggregate(x, by=list(ties), unique)$x)
w.ord <- cumsum(w) / sum(w)
}
cdf <- approxfun(w.ord, x, method = method, f = f, yleft = min(x), yright = max(x), rule = 2)
cdf(probs)

# Subject: Influence function of the trimmed mean (ipw)
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey

`.infltrim` <- function(x, weight, lo=0, hi=1){
  if(is.null(weight)){
    weight <- rep(1, length(x))
  }
xilo <- .svyquantiles(x, weight, probs=lo)
xhi <- .svyquantiles(x, weight, probs=hi)
below <- floor(lo * length(x))
above <- ceiling(hi * length(x))
mat <- c(rep((1 - lo) * xilo - (1 - hi) * xhi, below), rep(-lo * xilo - (1 - hi) * xhi, (above - below)),
          rep(hi * xhi - lo * xilo, (length(x) - above)))
if(below != 0){
x[1:below] <- 0
}
if(above != length(x)){
x[(above + 1):length(x)] <- 0
}
phi <- (x + mat) * (1/(hi - lo)) - .trimmedmean(x, weight, lo, hi)
return(phi)
}

# Subject: Influence function of the winsorized mean (ipw)
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey

`.inflwinsor` <- function(x, weight, lo=0, hi=1){
  if(is.null(weight)){
    weight <- rep(1, length(x))
  }
xilo <- .svyquantiles(x, weight, probs=lo)
xhi <- .svyquantiles(x, weight, probs=hi)
below <- floor(lo * length(x))
above <- ceiling(hi * length(x))
dens.lo <- .svydensity(x, weight, quant=lo)
dens.hi <- .svydensity(x, weight, quant=hi)
winlo <- lo - c(rep(1, below), rep(0, (length(x) - below)))
winhi <- hi - c(rep(1, above), rep(0, (length(x) - above)))
mat <- c(rep((1 - lo) * xilo - (1 - hi) * xihi, below),
rep(lo * xilo - (1 - hi) * xihi, (above - below)),
rep(hi * xihi - lo * xilo, (length(x) - above)))
if(below != 0){
x[1:below] <- 0}
if(above != length(x)){
x[(above + 1):length(x)] <- 0}
phi1 <- (x + mat) - .winsorizedmean(x, weight, lo, hi) * (hi - lo)
if(dens.lo == 0) {
phi <- phi1 + (1 - hi) * (winhi/dens.hi)
}
else{
phi <- phi1 + lo * (winlo/dens.lo) + (1 - hi) * (winhi/dens.hi)
}
return(phi)

# Subject: Estimation of the (ipw) density function at given quantiles
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey
#---------------------------------------------------------------
# INPUT
# x vector
# w survey weights
# quant vector of probabilities
# ngrid number of points at which the density is to be estimated
#---------------------------------------------------------------

function(x, weight, quant=0.5, ngrid=401) {
require(KernSmooth)
bwd <- dpik(x, scalest="minim", level=2, kernel="normal", canonical=FALSE, gridsize=ngrid,
range.x=range(x), truncate=TRUE)
at <- seq(min(x), max(x), length=ngrid)
xn <- rowsum(c(rep(0,ngrid),weight), c(1:ngrid, findInterval(x,at)))
dens <- locpoly(rep(1,ngrid),xn*ngrid/(diff(range(x))*sum(weight)), binned=TRUE,
bandwidth=bwd, range.x=range(x))
res <- dens$y[floor(length(dens$y)*quant)]
if(length(res)==0) res <- 0
return(res)
}

# Subject: SurveyCheck function: controls if the objects inherit the
# characteristics of the survey-package and checks if survey weights
# are zero
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey, Kernsmooth
#---------------------------------------------------------------
# INPUT
# x formula object
# design survey.design according to the "survey" package
# na.rm by default, the NA's will be removed
#---------------------------------------------------------------

function(x, design, na.rm=TRUE) {
require(survey)
if(!inherits(design, "survey.design2")){
stop("Design must be an object of the 'survey.design (2)' class\n")
}
method <- ""
if(design$call[[1]] == "subset"{
method <- paste(method, " (for domains)", sep="")
}
original.design <- design
if(!inherits(x, "formula")){
  stop("y must be a formula object\n")
} else{
  # ordering the data acording to the study variable
  # and generate a new svydesign with the ordered values
  ordereddata <- design$variables[order(design$variables[paste(x[2])]),]
  design$call$data <- substitute(ordereddata)
  design <- eval(design$call)
  # extract the variables from the svydesign -> xmat
  mf <- model.frame(x, design$variables, na.action=na.pass)
  xmat <- matrix(mrow = MROW(xx[[1]]), ncol = sum(cols))
  scols <- c(0, cumsum(cols))
  for (i in 1:length(xx)) {
    xmat[, scols[i] + 1:cols[i]] <- xx[i]
  }
  colnames(xmat) <- do.call("c", lapply(xx, colnames))
  xmat <- cbind(xmat, as.matrix(weights(design)))
  colnames(xmat)[NCOL(xmat)] <- "w"
}
# check if xmat has NAs
if(!is.na){
  nas <- rowSums(is.na(xmat))
  design <- design[nas == 0]
  xmat <- xmat[nas == 0, , drop=F]
  na <- length(which(nas==0))
} else{
  na <- NA
}
if(any(xmat[,NCOL(xmat)] == 0)) cat("There are weights=0\n")
res <- list(mat=xmat, method=method, na=na, design=design)
class(res) <- "rsvycheck"
return(res)

# Subject: IRLS function (iteratively re-weighted least squares)
# Author: Beat Hulliger Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey, robustbase
#-------------------------------------------------------------
# INPUT
# k robustness parameter
# mat matrix, columns: 1=y, 2=weights, 3=x
# acc numerical convergence criterion, e.g. 1e-5
# steps maximal irls-iterations
# design survey.design of the "survey" package
# beta.med initial beta-values
# mer boolean variable mer=TRUE for minimum est risk
# true.mean the "true" mean for the bias evaluation in mer
# psi psi-function object of "psi_func-class" (from robustbase)
#-------------------------------------------------------------
.irls <- function(k, mat, acc, steps, design, beta.med, mer=F, true.mean=NULL, psi){
  require(robustbase)
  niter <- 0
  beta.0 <- beta.med + 2*acc
  beta.r <- beta.med
  x.sqrt <- sqrt(mat[,3])
  while ((abs(beta.r - beta.0) > acc) & (niter < steps)){
    beta.0 <- beta.r
    stand.res <- (mat[,1] - beta.0 * mat[,3]) / x.sqrt
    scale <- .svyquantiles(abs(stand.res), mat[,2]) * 1.4829
    if (scale == 0) {
      scale <- .svyquantiles(abs(stand.res), mat[,2], probs=0.8) * 0.7803
      cat("MAD is 0, trying quantile 0.8 of absolute deviations\n")
    }
    beta.r <- beta.med + 2*acc
    #...
Chapter 6. RHT Package

```r
u <- psi@wgt(stand.res/scale, k)
beta.r <- sum(mat[,2]*u*mat[,1]) / sum(mat[,2]*u*mat[,3])
miter <= miter+1

ures <- u * (mat[,1] - beta.r * mat[,3])
design <- update(design, ures)
se <- as.vector(sum(mat[,2])*sqrt(vcov(svymean(-ures,design1))) / sum(mat[,2] * u * mat[,3]))
if(mer){
  mse <- as.vector(se^2 + (beta.r - true.mean)^2)
  return(mse)
} else{
  result <- list(m=beta.r, se=se, ures=ures, u=u, miter=miter, scale=scale)
  return(result)
}
```

# Subject: GUI - Status function: visual display of the computation progress
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: [none]

```r
status <- function(at, total){
  if(total >= 30)
    p.indicator <- diff(floor(30 * seq(1, total+1) / total))
    if(at==1){
      cat(" Progress \n")
      cat(" +------+------+------+------+ 
")
      cat(" 0% 25% 50% 75% 100% \n")
      cat(" ")
    } else{
      if(p.indicator[at]==1){
        cat("*
")
      } else{
      }
    }
  else{
    repeating <- diff(floor(seq(1, 30, length.out=total+1)))
    if(at==1){
      cat(" Progress \n")
      cat(" +------+------+------+------+ 
")
      cat(" 0% 25% 50% 75% 100% \n")
      for(j in 1:repeating[at]){  
        cat("*
")
      } else{
        for(j in 1:repeating[at]){  
          cat("*
")
        } }
    } if(at == total){
      cat("\n... done \n")
    }
  }
```

# Subject: Influence Function of the MAD
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Dependency: survey, Kernsmooth
```r
# INPUT
# x vector
# w survey weights

.inflmad <-
function(x, weight){
  T <- rht::svyquantiles(x, weight, probs=0.5)
  S <- rht::svyquantiles(abs(x - T), weight, probs=0.5)*1.4826
  density estimate at the median
  f <- rht::svydensity(x, weight, quant=0.5, ngrid=401)
  # quantile of T+S
  at.plus <- sum(weight[1:length(which(x <= T+S))])/sum(weight)
  # quantile of T-S
  at.minus <- sum(weight[1:length(which(x <= T-S))])/sum(weight)
  # density estimate at the quantile(T+S)
  f.plus <- rht::svydensity(x, weight, quant=at.plus, ngrid=401)
  # density estimate at the quantile(TSS)
  f.minus <- rht::svydensity(x, weight, quant=at.minus, ngrid=401)
  numerator <- sign(abs(x-T)-S) - ((f.plus - f.minus) / f)*sign(x-T)
  denominator <- 2*(f.plus + f.minus)
  res <- as.vector(numerator/denominator)
}
```

```r
# INPUT
# x vector
# weight survey weights
# T M-estimate
# k tuning constant of the M-estimator
# S MAD-estimate
# psi psi-function
# exact TRUE=considering effect of preliminary MAD; else ignoring

.inflmest <- function(x, weight, T, k, S, psi = huberPsi, exact = T){
  require(robustbase)
  u <- (x-T)/S
  psi.value <- psi0psi(u, k)
  psiprime.value <- psi0Dpsi(u, k)
  if(exact){
    numerator <- psi.value*S - .inflmad(x, weight)*sum(weight*u*psiprime.value))/(sum(weight))
  } else{
    numerator <- psi.value
  }
  denominator <- sum(weight*psiprime.value)/sum(weight)
  res <- numerator/denominator
  return(res)
}
```

```r
# Subject: Asymmetric Huber Psi Function
# Object of the "Psi Function"-Class in "robustbase"
# Author: Tobias Schoch
# Status: v1.1 (mature)
# Date: February 8 2009
# Depends: robustbase

asymhuberPsi <- psiFunc(
  rho = function(x, k=1.345){
    r <= u < x
    I <= u < k
    r[I] <= u[I]-2/2
    r[I] <= k * (u[I] - k/2)
  
  ```
Chapter 6. RHT Package

```r
psi = function(x, k = 1.345) {
  pmin(k, x),
},

wgt = function(x, k = 1.345) {
  pmin(1, k/x),
},

Dpsi = function(x, k = 1.345) {
  x <= k,
},

Erho = function(k) k,
Epsi2 = function(k) k,
EDpsi = function(k) k
```

# Subject: Huber Psi Function
# Object of the "Psi Function"-Class in "robustbase"
# Author: Martin Maechler, with modifications by Tobias Schoch
# Mod: 2nd argument in the pmin2 function (is called by huberPsi@wgt)
# cannot deal with inf (results because k/abs(r) is inf when
# residual r is exactly zero)
# Status: v1.1 (mature)
# Date: February 8 2009
# Depends: robustbase
Chapter 7
Robust Distribution Fitting

```r
fitLN <- function(x) {
  # ----------------------------------------
  # Authors: Josef Holzer and Andreas Alfons
  # Vienna University of Technology
  # Licence: GPL 2.0
  # ----------------------------------------
  ## initializations
  ind <- 1:length(x)
  if(any()) ind <- ind[!i]
  ord <- order(x[ind])
  ind <- ind[ord]  # indices of sorted vector
  ## obtain parameters of lognormal distribution
  m <- mean(x[ind])
  v <- var(x[ind])
  vlog <- v/m^2 + 1
  meanlog <- log(m) - vlog/2  # mean on the log scale
  sdlog <- sqrt(vlog)  # standard deviation on the log scale
  ## fit lognormal distribution
  x[ind] <- sort(rlnorm(length(ind), meanlog, sdlog))
}
```

```r
fitPareto <- function(x, k, method = "thetaPDC", group = NULL, ...) {
  # ----------------------------------------
  # Authors: Josef Holzer and Andreas Alfons
  # Vienna University of Technology
  # ----------------------------------------
  ## check for missing values
  indices <- 1:length(values)
  values <- x[indices]
  # initializations
  if(!is.numeric(x) || length(x) == 0) stop("'x' must be a numeric vector")
  if(!is.numeric(k) || length(k) == 0 || k[1] < 1) {
    stop("'k' must be a positive integer")
  }
  else k <- k[1]
  if(is.null(group)) values <- x
  else {
    if(is.vector(group) && !is.factor(group)) {
      stop("'group' must be a vector or factor")
    }
    if(length(group) != length(x)) {
      stop("'group' must have the same length as 'x'")
    }
    if(any(is.na(group))) stop("'group' contains missing values")
    unique <- !duplicated(group)
    values <- x[unique]
  }
  # check for missing values
  indices <- 1:length(values)
```

© http://ameli.surveystatistics.net/ - 2011
if(any(i <= is.na(values))) indices <- indices[i]
## order of observed values
order <- order(values[indices])
indicesSorted <- indices[order]  # indices of sorted vector
n <- length(indicesSorted)
if(k >= n) stop("'k' must be smaller than the number of observed values")
xm <- values[indicesSorted[n-k]]  # threshold (location parameter)
## estimate shape parameter
theta <- do.call(method, list(values[indices], k, ...))
## fit Pareto distribution
valuesPareto <- xm / runif(k)^(1/theta)
values[indicesSorted[(n-k+1):n]] <- valuesPareto
values[indicesSorted[(n-k+1):n]] <- sort(valuesPareto)
## return values
if(!is.null(group)) {
  group <- as.character(group)
  names(values) <- group[unique]
  values <- values[group]
  names(values) <- names(x)
}
values

# ----------------------------------------
# Authors: Josef Holzer and Andreas Alfons
# Vienna University of Technology
# ----------------------------------------
thetaWML <- function(x, k, weight = c("residuals", "probability"),
const, bias = TRUE, tol = 1e-5, ...) {
  ## initializations
  if(!is.numeric(x) | length(x) == 0) stop("'x' must be a numeric vector")
  if(!is.numeric(k) | length(k) == 0 | k[1] < 1) {
    stop("'k' must be a positive integer")
  } else k <- k[1]
  if(any(i <= is.na(x))) x <- x[i]  # remove missing values
  x <- sort(x)
  n <- length(x)
  if(k >= n) stop("'k' must be smaller than the number of observed values")
  x0 <- x[n-k]  # threshold (location parameter)
  xt <- x[(n-k+1):n]  # tail (values larger than threshold)
  y <- log(xt/x0)  # relative excesses
  weight <- match.arg(weight)
  ## define weight function and function for root finding
  if(weight == "residuals") {
    ## check tuning constant
    if(missing(const)) const <- 2.5
    else if(!is.numeric(const) | length(const) == 0) {
      stop("'const' must be a numeric value")
    } else const <- const[1]
    ## some temporary values
    h <- k:1
    by <- log(h/(k+1))
    hsig <- sqrt(cumsum(1/h^2))
    ## objective function
    zeroTheta <- function(theta) {
      r <- (theta*y + by) / hsig  # standardized residuals
      w <- pmn(1, const/abs(r))  # weights
      ## derivative of log(f) with respect to theta: 1/theta - log(xt/x0)
      sum(w * (1/theta - y))
    }
  } else {
    ## check tuning constants
    if(missing(const)) const <- rep.int(0.005, 2)
    else if(!is.numeric(const) | length(const) == 0) {
      stop("'const' must be a numeric vector of length two")
    } else const <- rep(const, length.out=2)
    p1 <- const[1]
    p2 <- const[2]
## objective function

```r
zeroTheta <- function (theta) {
  F <- 1 - (xt/x0)^(-theta) # distribution function
  w <- ifelse(F < p1, F/p1, ifelse(F <= 1-p2, 1, (1-F)/p2)) # weights
  sum(w * (1/theta - y))
}
```

## solving sum(\phi(xt,theta))=0

```
## derivative of log (f) with respect to theta : 1/theta - log (xt/x0)

## minimization of quadratic function instead of root finding

```r
localNlm <- function(f, p = NULL, tol, ...) {
  if(is.null(p)) p <- thetaHill(x, k) # starting parameter
  theta <- nlm(function(theta) f(theta)^2, p, ...)$estimate
  if(abs(f(theta)) < tol) theta
  else {
    warning("M- estimator for 'theta' could not ",
           "be obtained, starting value is returned")
    p
  }
}
theta <- localNlm(zeroTheta, tol=tol, ...)
```

## optional bias correction

```
if(bias) {
  if(weight == "residuals") {
    f <- (theta+y + hy) / hsig # standardized residuals
    w <- pmin(1, const/abs(f)) # weights
    F <- 1 - (xt/x0)^(-theta) # distribution function
    deltaF <- diff(c(0, F)) # difference operator applied to F
    dlogf <- 1/theta - y # derivative of log(f)
    d2logf <- -1/theta^2 # second derivative of log(f)
    # derivative of weight function
    dw <- ifelse(w == 1, 0, (-const) * y * hsig / (theta*y + hy)^2)
    # bias correction
    bcorr <- sum(w*dlogf*deltaF)/sum((dw*dlogf + w*d2logf) * deltaF)
  } else {
    cp1 <- 1-p1
cp2 <- 1-p2
    # bias correction
    bcorr <- ((theta/2) *
               (2*cp1^2*log(cp1) + pl*cp1 + pi*cp2 + 2*pi*cp2*log(p2)) /
               ((cp1*log(cp1))'' - pi*cp1 - pi*cp2 + pi*cp2*(log(p2))''2)
  }
  # apply bias correction to theta
  theta <- theta - bcorr
}

## return WML estimate
theta
```
ISE <- function(\theta, y, k) {
  f <- \theta * y^{(-1 - \theta)}
  sumf <- sum(f)
  pf2 <- \theta^2/(2*\theta + 1) # primitive of f^2
  u <- (1/k*sumf)/pf2
  w <- (1/k*sumf)/pf2
  w^2 - 2*u/k*sumf
}

## minimize
localNlm <- function(f, p = NULL, ...) {
  if(is.null(p)) p <- thetaHill(x, k) # starting parameter
  nlm(f, p, ...)
}

localNlm(ISE, y=y, k=k, ...)$estimate
Chapter 8

Robustification of the QSR

# BQSR <- function(x, design, lower=0, upper=0) {
  r <- 0.8 # rich quintile
  p <- 0.2 # poor quintile
  require(survey, rht)
  weight <- weights(design)
  x.variable <- as.vector(model.matrix(x, design$variables, na.action = na.pass)[,2])
  # compute the means
  overall.mean <- rht:::.trimmedmean(x.variable, weight, 0, 1-upper)
  rich.mean <- rht:::.trimmedmean(x.variable, weight, 0, r)
  poor.mean <- rht:::.trimmedmean(x.variable, weight, 0, p-lower)
  # compute the influence function (linearization value)
  overall.z <- .inflqsm(x.variable, weight, 1-upper)
  rich.z <- .inflqsm(x.variable, weight, r)
  poor.z <- .inflqsm(x.variable, weight, p-lower)
  # compute the linearized ratio
  z <- (((1-upper)*overall.z - r*rich.z)/(p-lower)*poor.mean - ((1-upper)*overall.mean - r*rich.mean)*((p-lower)*poor.z))/((p-lower)*poor.mean)^2
  design <- update(design, lin=z)
  sigma <- vcov(svymean(~ lin, design))
  estimate <- (((1-upper)*overall.mean - r*rich.mean)/(p-lower)*poor.mean)
  attr(estimate, "") <- 
  attr(estimate, "var") <- sigma
  attr(estimate, "statistic") <- "BQSR"
  attr(estimate, "method") <- "BQSR"
  class(estimate) <- c("svystat.rob")
  return(estimate)
}
z <- xi - qsm + (1/q)*(x-xi)*(x <= xi)
return(z)
Chapter 9

MODI Package

# BEM Algorithm for Multivariate Outlier Detection in Incomplete Multivariate Normal Data
#
# BEM algorithm as described in:
# BEGUIN, C. and HULLIGER, B. (2008)
# The BACON-EEM Algorithm for Multivariate Outlier Detection
# in Incomplete Survey Data, Survey Methodology, Vol. 34, No. 1, pp. 91-103.
#
# BACON approach as described in:
# BACON: Blocked Adaptive Computationally-Efficient
# Outlier Nominators, Computational Statistics and
# Analysis, in press.
#
# EM approach as described in:
# Schafer J.L. (2000),
# Analysis of Incomplete Multivariate Data,
# Monographs on Statistics and Applied Probability 72,
# Chapman & Hall.
#
# Program by CEDRIC BEGUIN
# Last modified : 20 March 2009 Beat Hulliger
# Copyright : Swiss Federal Statistical Office and University of Neuchatel, 2002
# Translated to R by Beat Hulliger, 9 May 2003, 22 November 2004
# 20.3.09: weighted.var replaced by cov.wt(,, method = "ML"),
# weighted.var.na replaced by external weighted.var (analogue to weighted.mean)
# EM.normal function external
#
# ################# BEM ##################
# Main program: will use EM for each computation of a location and scatter estimate when
# items are missing.
BEM <- function(data, weights, v=2, c0=3, alpha=0.01, md.type="m", em.steps.start=10, em.steps.loop=5, better.estimation=F, steps.output=F)
{

# Preprocessing of the data
if (!is.matrix(data)) data<-as.matrix(data)
if (missing(weights)) weights<-rep(1,n)

# Removing the unit(s) with all items missing
new.indices <- which(apply(is.na(data),1,prod)==0)
discarded<-NA
nfull<-n
if (length(new.indices)<n)
{

© http://ameli.surveystatistics.net/ - 2011
Chapter 9. MODI Package

discarded <- which(apply(is.na(data),1,prod)==1)
cat("Warning: missing observations",discarded,"removed from the data\n")
data <- data[new.indices,]
weights <- weights[new.indices]
n <- nrow(data)
}
if (steps.output) cat("End of preprocessing\n")

# Computation time start

calc.time <- proc.time()

# Order the data by missingness patterns:
s.patterns <- vector of length n, with the missingness patterns stocked as strings of the type "11010...011" with "1" for missing.
data, weights and s.patterns ordered using s.patterns' order
s.patterns <- apply(matrix(as.integer(is.na(data)),n,p),1,paste,sep="",collapse="")
perm <- order(s.patterns)
data <- data[perm,]
s.patterns <- s.patterns[perm]
weights <- weights[perm]

# Missingness patterns stats:
s.counts <- as.vector(table(s.patterns))
s.id <- cumsum(s.counts)
S <- length(s.id)
missing.items <- is.na(data[s.id,,drop=F])
nb.missing.items <- apply(missing.items,1,sum)

# Constants used by the BACON algorithm to select the good points

N <- sum(weights)
initial.length <- c0*p
if (initial.length>n) stop("nInitial length bigger than number of observations. Please, decrease c0.")
c.np <- 1 + (p+1)/(N-p) + 2/(N-1-3*p)
h <- floor((N+p+1)/2)
if (alpha<0.5) alpha<-1-alpha
chi.sq <- qchisq(alpha,p)

# Step 1

The two possible starts for BACON, modified to deal with missing items:
Version 2 (default): the distances are the Euclidean distances from the componentwise median;
the median is computed by removing the missing items in each variable;
missing values in an observation are not included in the distance to the median.
If pg is the number of columns in which no missing values occur for that observation,
then the distance returned is sqrt(p/pg) times the
Euclidean distance between the two vectors of length pg shortened to exclude missing values.
Version 1: the usual mean and covariance matrix are used to compute Mahalanobis distances;
both mean and covariance are computed by EM if any missing value occurs;
the Mahalanobis distance for an observation is computed by restricting the mean and the covariance matrix to the subspace of non-missing variables and
the sqrt(p/pg) correction is used as in Version 1.

if (v==2) {

AMELI-WP4-D4.1
# Version 2

EM.mean <- apply(data,2,weighted.quantile,w=weights)
dist <- apply(sweep(data,2,EM.mean)^2,1,sum,na.rm=T)*p/(p-apply(is.na(data),1,sum))

# Version 1

if (S==1 & nb.missing.items[1]==0)
{
  # Case where no missing value occurs => regular mean and covariance matrix
  EM.mean <- apply(data,2,weighted.mean,w=weights)
  EM.var <- cov.wt(data,wt=weights,center=EM.mean,method="ML")$cov
}
else
{
  # Case where missing values occur => mean and covariance matrix computed by EM
  T.obs <- matrix(0,p+1,p+1)
  if (nb.missing.items[1]==0)
  {
    # Case where some observations are complete (no missing item) => computation of
    # the sufficient statistics on these observations and then EM.
    if (steps.output) cat("Preparation of T.obs for version 1\n")
    weights.obs <- weights[1:s.counts[1]]
    T.obs[1,] <- T.obs[,1] <- c(sum(weights.obs),apply(weights.obs*data[1:s.
      counts[1],],2,sum))
    for (i in 1:s.counts[1])
    {
      T.obs[2:(p+1),2:(p+1)] <- T.obs[2:(p+1),2:(p+1)]+weights.obs[i]*data[
        i,]%*%t(data[i,])
    }
    EM.result <- EM.normal(data=data[(s.id[1]+1):n,drop=F],weights=weights[
      s.id[1]+1:n],n=N,p=p,
      s.counts=s.counts[2:S],s.id=s.id[2:S]-s.id [1],S=S-1,T.obs=T.obs,
      start.mean=apply(data,2,weighted.mean,w=weights,na.rm=TRUE),
      start.var=diag(apply(data,2,weighted.var,w=weights,na.rm=TRUE)),numb.it=em.steps.
      start,
      Estep.output=steps.output)
  } else
  {
    # Case where all observations have missing items => EM
    EM.result <- EM.normal(data=data,weights,n=N,p=p,
      s.counts=s.counts,s.id=s.id,S=T.obs,T.obs,stop.mean=apply(data,2,weighted.mean,w=weights,na.rm=TRUE),
      start.var=diag(apply(data,2,weighted.var,w=weights,na.rm=TRUE)),numb.it=em.steps.
      start,
      Estep.output=steps.output)
  }
EM.mean <- EM.result[1,2:(p+1)]
EM.var <- EM.result[2:(p+1),2:(p+1)]

# Computation of the Mahalanobis distances
# indices <- (!missing.items[i,])
if (md.type == "c") EM.var.inverse <- solve(EM.var) else EM.var.inverse <- EM.var
dist <- mahalanobis(data[1:s.id[1],indices,drop=F],
                   EM.mean[indices],EM.var.inverse[indices,indices],
                   inverted=(md.type == "c"))*p/(p-nb.missing.items[i])
if (S>1)
{
  for (i in 2:S)
  {
    indices <- (!missing.items[i,])
    dist <- c(dist,mahalanobis(data[(s.id[i-1]+1):s.id[i],indices,drop=F),
                      EM.mean[indices],EM.var.inverse[indices,indices,drop=F],
                      inverted=(md.type == "c"))*p/(p-nb.missing.items[i])
  }
}

if (steps.output) cat("Version ",v,"": estimation of mean = 
                  signif(EM.mean,4),"\n")

# Selection of the initial basic good subset using the computed distance
ordre <- order(dist)
good <- oder[1:initial.length]
good <- good[order(good)]
if (steps.output) cat("Initial good subset size: ",length(good),"\n")

# Statistics of the missingness patterns of the good subset
n.good <- length(good)
s.patterns.good <- s.patterns[good]
s.counts.good <- as.vector(table(s.patterns.good))
s.id.good <- csum(s.counts.good)
S.good <- length(s.id.good)
missing.items.good <- is.na(data[good,s.id.good],drop=F)
nb.missing.items.good <- apply(missing.items.good,1,sum)
weights.good <- weights[good]
N.good <- sum(weights.good)

# Determination of the indices (if any) of the observations in the good subset
# without missing items
T.obs.good.exist <- (nb.missing.items.good[1]==0)
if (T.obs.good.exist)
{
  T.obs.good.indices <- good[1:s.id[1]]
}

# Computation of mean and covariance matrix of the good subset by EM
if (S.good==1 & T.obs.good.exist)
{
  # Case where no missing value occurs => regular mean and covariance matrix computed
  # and the sufficient
  # statistics deduced from them
  EM.mean.good <- apply(data[good,],2,weighted.mean,v=weights.good)
  EM.var.good <- cov.wt(data[good,],wt=weights.good,center=EM.mean.good,method="ML")$cov
  T.obs.good <- matrix(0,p+1,p+1)
  T.obs.good[1,] <- T.obs.good[1,] <- c(-1,EM.mean.good)
  T.obs.good[2:(p+1),2:(p+1)] <- EM.var.good*(N.good-1)/N.good
  T.obs.good <- .sweep.operator(T.obs.good,1,TRUE)*N.good
  T.obs.good.exist <- T
}
else
{
  # Case where missing values occur => mean and covariance matrix computed by EM
  T.obs.good <- matrix(0,p+1,p+1)
if (T.obs.good.exist) {
    # Case where some observations are complete (no missing item) => computation of
    # the sufficient statistics on these observations and then EM.
    if (steps.output) cat("Preparation of T_obs\n")
    weights.good.obs <- weights.good[1: s.counts.good[1]]
    T.obs.good[1,] <- c(sum(weights.good.obs),
        apply(weights.good.obs * data[1: s.counts.good[1]], 2, sum)
    )
    for (i in 1: s.counts.good[1])
        T.obs.good[2:(p +1),2:(p +1)] <- T.obs.good[2:(p +1),2:(p +1)]
            + weights.good.obs[i] * data[i,]
    EM.result.good <- EM.normal(data=data[good[1: s.counts.good[1]],
        drop=F],
        weights=weights.good[1: s.counts.good[1]],
        n=N
        good,p=p,
        s.counts=s.counts.good[2:S.good],
        s.id=s.id.good[2:S.good]-s.id.good[1],
        S=S.good-1,T.obs=T.obs.good,
        start.mean=apply(data[good[,]],2,weighted.mean,
        w=weights.good,na.rm=TRUE),
        start.var=diag(apply(data[good[,]],2,weighted.var,
        w=weights.good,na.rm=TRUE)),
        numb.it=em.steps.loop,Estep.output=steps.output)
} else {
    # Case where all observations have missing items => EM
    EM.result.good <- EM.normal(data=data[good,,
        drop=F],weights=weights.good,n=N.
        good,p=p,
        s.counts=s.counts.good,s.id=s.id.good,S=S.
        good,T.obs=T.obs.good,
        start.mean=apply(data[good[,]],2,weighted.mean,
        w=weights.good,na.rm=TRUE),
        start.var=diag(apply(data[good[,]],2,weighted.
        var,w=weights.good,na.rm=TRUE)),
        numb.it=em.steps.loop,Estep.output=steps.output)
    EM.mean.good <- EM.result.good[1,2:(p +1)]
    EM.var.good <- EM.result.good[2:(p +1),2:(p +1)]
} if (steps.output) cat("First good subset estimation of mean = ",signif(EM.mean.good
        ,4),"\n")
# Test if the size of the good subset is not too small (singular covariance matrix)
# if (qr(EM.var.good)$rank < p)
#     stop("Initial subset size too small, please increase c0")
#
# Main loop of BACON: increase the good subset using the Mahalanobis distances computed
# from it;
# the Mahalanobis distances are computed as in Version 1 of the start (see above).
#
# count <- 0
# repeat
#     count <- count+1
#     # Upgrade of the constants values used by BACON
#     r <- sum(weights.good)
#     c.hr <- max(0,(h-r)/(h+r))
#     test <- (c.np+c.hr)^2*chi.sq
# Computation of the Mahalanobis distances

```r
indices <- (!missing.items[1,])
if (md.type == "c") EM.var.good.inverse <- solve(EM.var.good)
else EM.var.good

dist <- mahalanobis(data[1:s.id[1], indices, drop=F], EM.mean.good[indices], EM.var.
    good.inverse[indices, indices], inverted=(md.type=="c"))*p/(p-nb.missing.items
    [1])
if (S>1)
{
    for (i in 2:S)
    {
        indices <- (!missing.items[i,])
        dist <- c(dist, mahalanobis(data[(s.id[i-1]+1):s.id[i], indices, drop=F], EM.
            mean.good[indices], EM.var.good.inverse[indices, indices, drop=F],
            inverted=(md.type=="c"))*p/(p-nb.missing.items[i])
    }
}
```

# Memorization of some data on the preceding good subset

```r
oldgood <- good
T.obs.oldgood.exist <- T.obs.good.exist
if (T.obs.oldgood.exist)
{
    T.obs.oldgood.indices <- T.obs.good.indices
}
```

# Determination of the new good subset

```r
good <- (1:n)[dist<=test]
if (steps.output) cat("Loop ", count, ": start; test=",test,"; good subset=",length
    (good),"\n")
```

# Comparison with the preceding good subset, break if equality

```r
if (length(good)==length(oldgood))
    if (prod(good==oldgood)) break
```

# Statistics of the missingness patterns of the new good subset

```r
n.good <- length(good)
s.patterns.good <- s.patterns[good]
s.counts.good <- as.vector(table(s.patterns.good))
s.id.good <- cumsum(s.counts.good)
S.good <- length(s.id.good)
missing.items.good <- is.na(data[good[s.id.good,, drop=F])
nb.missing.items.good <- apply(missing.items.good,1,sum)
weights.good <- weights[good]
N.good <- sum(weights.good)
```

# Determination of the indices (if any) of the observations in the new good
# subset without missing items

```r
T.obs.good.exist <- (nb.missing.items.good[1]==0)
if (T.obs.good.exist)
{
    T.obs.good.indices <- good[1:s.id.good[1]]
}
```

# Computation of mean and covariance matrix of the new good subset

```r
if (S.good==1 & T.obs.good.exist)
{
    # Case where no missing value occurs => regular mean and covariance matrix
    # computed and the sufficient statistics deduced from them
    EM.mean.good <- apply(data[good,,2,weighted.mean,w=weights.good])
```
EM.var.good <- cov.wt(data[, good], wt=weights.good, center=EM.mean.good, method="ML") & cov
T.obs.good <- matrix(0,p+1,p+1)
T.obs.good[1,] <- T.obs.good[,1] <- c(-1, EM.mean.good)
T.obs.good[2:(p+1),2:(p+1)] <- EM.var.good*(N.good-1)/N.good
T.obs.good <- sweep.operator(T.obs.good,1,TRUE) * N.good
T.obs.good.exist <- T
}
else {
  # Case where missing values occur => mean and covariance matrix computed by EM with
  # starting parameters set to the preceding estimates
  if (T.obs.good.exist)
    {
      # Case where some observations are complete (no missing item) => computation of
      # the sufficient statistics on these observations and then EM with starting
      # parameters
      # set to the preceding estimates
      if (T.obs.oldgood.exist)
        {
          # Case where sufficient statistics from complete data were computed in
          # the preceding good subset => upgrade of these statistics substracting the
          # observations that were deleted from the preceeding good subset and adding the observations
          # that were added to it
          if (steps.output) cat("Updating of T_obs\n")
          good.boo <- oldgood.boo <- rep(F,n)
          good.boo[T.obs.good.indices] <- T

          for (i in (1:n)[xor(good.boo,oldgood.boo)&oldgood.boo])
            {
              T.obs.good[1,] <- T.obs.good[1,] - weights.good.obs[i]*c(1, data[i,])
              T.obs.good[2:(p+1),2:(p+1)] <- T.obs.good[2:(p+1),2:(p+1)] -
                                          weights.good.obs[i]*data[i,]%*%t(data[i,])
            }
          for (i in (1:n)[xor(good.boo,oldgood.boo)&good.boo])
            {
              T.obs.good[1,] <- T.obs.good[1,] + weights.good.obs[i]*c(1, data[i,])
              T.obs.good[2:(p+1),2:(p+1)] <- T.obs.good[2:(p+1),2:(p+1)] +
                                          weights.good.obs[i]*data[i,]%*%t(data[i,])
            }
          T.obs.good[,1] <- T.obs.good[1,]
        }
      else
        {
          # Case where no sufficient statistics from complete data were computed in
          # the preceeding good subset => computation of these statistics for the new good subset
          if (steps.output) cat("New preparation of T_obs\n")
          T.obs.good <- matrix(0,p+1,p+1)
          weights.good.obs.1 <- weights.good[1:s.counts.good[1]]
          T.obs.good[1,] <- T.obs.good[,1] <- c(sum(weights.good.obs), apply(weights.good.obs*data[good[1:s.counts.good[1]], drop=F],2,sum))
          for (i in 1:s.counts.good[1])
            {
              T.obs.good[2:(p+1),2:(p+1)] <- T.obs.good[2:(p+1),2:(p+1)] -
                                          weights.good.obs[i]*data[good[i,]]%*%t(data[good[i,]])
            }
        }
  }

© http://ameli.surveystatistics.net/ - 2011
EM.result.good <- EM.normal(data=data[good[(s.id.good[1]+1):n.good]], drop =F), weights=good[(s.id.good[1]+1):n.good], n=N.good, p=p, s.counts=s.counts.good[2:S.good], s.id=s.id.good[2:S.good], S=S.good-1, T.obs=T.obs.good, start.mean=EM.mean.good, start.var=EM.var.good, numb.it=em.steps.loop, Estep.output=steps.output)
}
else{
  # Case where all observations have missing items => EM with starting parameters set to the preceding estimates
  T.obs.good <- matrix(0,p+1,p+1)
  EM.result.good <- EM.normal(data=data[good,], drop=F), weights=good, n=N.good, p=p, s.counts=s.counts.good, s.id=s.id.good, S=S.good, T.obs=T.obs.good, start.mean=EM.mean.good, start.var=EM.var.good, numb.it=em.steps.loop, Estep.output=steps.output)
}
EM.mean.good <- EM.result.good[1,2:(p+1)]
EM.var.good <- EM.result.good[2:(p+1),2:(p+1)]
}
# Check if the computed covariance is singular, stop in that case.
if (steps.output) cat("Loop ",count," end: estimation of mean = ",signif(EM.mean.good,4),"\n")
if (qr(EM.var.good)$rank < p) stop("Singular covariance matrix with a particular subset (try a bigger alpha).")
}

~~~~~~~~~~~ Computation time stop ~~~~~~~~~~~
# calc.time <- proc.time() - calc.time

~~~~~~~~~~~~~~~ Step 5 ~~~~~~~~~~~~~~~
# Nominate the outliers using the original numbering (without discarded obs.)
outliers <- perm[(1:n)[-good]]

# If a better estimation is seeked, "em.steps.start" more steps of EM are taken if (better.estimation) {
  if (T.obs.good.exist)
    {
      # Case where some observations are complete, more steps of EM with the sufficient statistics computed before
      EM.result.good <- EM.normal(data=data[good[(s.id.good[1]+1):n.good]], drop=F), weights=good[(s.id.good[1]+1):n.good], n=N.good, p=p, s.counts=s.counts.good[2:S.good], s.id=s.id.good[2:S.good], S=S.good-1, T.obs=T.obs.good, start.mean=EM.mean.good, start.var=EM.var.good, numb.it=em.steps.start, Estep.output=steps.output)
    }
  else{
    # Case where all observations have missing items => more steps of EM
    EM.result.good <- EM.normal(data=data, drop=F), weights=good, n=N.good, p=p, s.counts=s.counts.good, s.id=s.id.good, S=S.good, T.obs=T.obs.good,
\begin{verbatim}
start.mean=EM.mean.good,start.var=EM.var.good ,

numb.it=em.steps.start,Estep.output=steps.output)
}
EM.mean.good <= EM.result.good[1,2:(p+1)]
EM.var.good <= EM.result.good[2:(p+1),2:(p+1)]

# Computation of the Mahalanobis distances
# indices <= (!missing.items[i,])
if (md.type=="c") EM.var.good.inverse <- solve(EM.var.good) else EM.var.good.

dist <= mahalanobis(data[1:s.id[1],indices,drop=F],EM.mean.good[indices],EM.var.
good.inverse[indices,indices],inverted=(md.type=="c"))*p/(p-nb.missing.items
[1])

# distances in the original numbering

dist<dist[order(perm)]
cutpoint<min(dist[outliers])
# outliers and distances in original numbering with full dataset

outn<=logical(n)
outn[outliers]<-TRUE
outnfull<=logical(nfull)
outnfull[new.indices]<-outn
distnfull<=rep(NA,nfull)
distnfull[new.indices]<-dist

# *************** Results ***************

BEM.r <= list(sample.size = n,
discarded.observations=discarded,
number.of.variables = p,
significance.level = alpha,
initial.basic.subset.size = initial.length ,
final.basic.subset.size = length(good),
number.of.iterations = count,
computation.time = calc.time,
center = EM.mean.good ,
scatter = EM.var.good ,
cutpoint=cutpoint)

BEM.i <= list(outind = outnfull,dist = distnfull)

# *************** Output ***************

cat("\n","BEM has detected",length(outliers)," outlier(s) in",calc.time," seconds. ","\n","\n")
cat(" The results are in BEM.r and BEM.i","\n")
\end{verbatim}
### Initial tests ###

- **c**: Tuning constant for redefinition of outliers
- **beta**: Condition for link to donor
- **alpha**: Weight of failing items

---

**POEM** function (data, weights, outind, errors, missing.matrix, alpha = 0.5, beta = 0.5, reweight = FALSE, out = FALSE, c = 5, preliminary.mean.imputation = FALSE, verbose = FALSE)

The function is designed for multivariate weighted imputation for outliers, edit failure and missing values. It is described in:

- BEGUIN, C., HULLIGER, B., (2002),
- EUREDIT Workpackage x.2 D4-5.2.1-2. C
- Develop and evaluate new methods for statistical outlier detection and outlier robust multivariate imputation,

#### Program by CEDRIC BEGUIN

Last modified: 7 August 2009 (Beat Hulliger)

### Technical Details

```r
POEM <- function(data, weights, outind, errors, missing.matrix, alpha = 0.5, beta = 0.5, reweight = FALSE, out = FALSE, c = 5, preliminary.mean.imputation = FALSE, verbose = FALSE) {
  # POEM Algorithm for multivariate weighted imputation for Outliers, Edit failure and Missing values
  # POEM algorithm as described in:
  # BEGUIN, C., HULLIGER, B., (2002),
  # EUREDIT Workpackage x.2 D4-5.2.1-2. C
  # Develop and evaluate new methods for statistical outlier detection and outlier robust multivariate imputation,
  # Program by CEDRIC BEGUIN
  # Last modified: 7 August 2009 (Beat Hulliger)
  # Copyright: Swiss Federal Statistical Office, 2002
  # alpha: Weight of failing items
  # beta: Condition for link to donor
  # c: Tuning constant for redefinition of outliers
  # Initial tests ###

  if (!is.matrix(data)) data <- as.matrix(data)
  n <- nrow(data)
  p <- ncol(data)

  if (missing(weights)) weights <- rep(1, n)
  if (is.logical(outind)) outind <- as.numeric(outind)
  if (missing(missing.matrix)) missing.matrix <- (1 - is.na(data))
  if (missing(errors)) errors <- matrix(1, nrow = n, ncol = p)
  if (!is.vector(weights)) stop("Weights not in vector form","\n")
  if (!is.matrix(missing.matrix)) stop("Missing values not in matrix form","\n")
  if (!is.vector(outind)) stop("Outind not in vector form","\n")
  if (!is.matrix(errors)) stop("Errors not in matrix form","\n")
  if (length(weights) != n | ncol(missing.matrix) != p) stop("Missing values matrix do not have same dimensions as data","\n")
  if (length(outind) != n) stop("Wrong length of outind","\n")
  if (nrow(errors) != n | ncol(errors) != p) stop("Errors matrix do not have same dimensions as data","\n")
  if (sum(is.na(weights)) > 0) stop("Missing values in weights","\n")
  if (sum(is.na(missing.matrix)) > 0) stop("Missing values in missing data matrix","\n")
  if (sum(is.na(outind)) > 0) stop("Missing values in outind","\n")
  if (sum(is.na(errors)) > 0) stop("Missing values in errors matrix","\n")
}
```

---

**POEM Algorithm for multivariate weighted imputation for Outliers, Edit failure and Missing values**

1. **c**: Tuning constant for redefinition of outliers
2. **beta**: Condition for link to donor
3. **alpha**: Weight of failing items

# Completely missing
comp.miss <- which(apply(is.na(data),1,prod)==1)
cat("\nNumber of completely missing observations ",sum(comp.miss),"\n")

########### Computation time start ############

# Set all missing values to zero ###
old.data <- data
data[!(missing.matrix)] <- 0

### Computation of alpha_ij ###
alpha.ij <- missing.matrix*alpha^(1-errors)
good.values <- apply(weights*alpha.ij,2,sum)
old.number.of.nonoutliers <- sum(1-outind)
old.weighted.sum.of.nonoutliers <- sum(weights*(1-outind))
missing.errors <- missing.matrix*errors

### Computation of center ###


### Centering of data ###
data <- sweep(data,2,center,"-")

### Computation of coordinates variances ###
variances <- apply((1-outind)*weights*alpha.ij*data^2,2,sum)
if (sum(variances==0)>0)
{
  zero.variances <- which(variances==0)
  cat("Warning: Variable(s)",zero.variances,"has (have) zero variance(s)\n")
  stop("\nRemove these variables or reduce the set of outliers\n")
}

### Standardization of data ###
data <- sweep(data,2,sqrt(variances),"/")

### Computation of covariance matrix ###
covariance <- (t(alpha.ij*data)*((1-outind)*weights*alpha.ij*data))
if (!preliminary.mean.imputation)
{
  # Formula (12) on page 121 of the Euredit Deliverable
  covariance <- covariance/(t(alpha.ij)*((1-outind)*weights*alpha.ij))
  if (determinant(covariance)$sign<0)
  {
    cat("Warning: Covariance matrix not positive definite with original data including
          missing values\n")
    cat("Choose option preliminary.mean.imputation=T!\n")
  }
  # preliminary mean imputation is Formula (13) on page 122 of the Euredit Deliverable.
  # This is equivalent to setting tilde x to zero. And this in turn
  # is equivalent to setting x to the mean before standardising.
  else covariance <- covariance / sum((1-outind)*weights)
  if (verbose) {cat("Covariance matrix of standardised observations\n")
    print(covariance)
  }
}

### Reweighting of outliers ###

MD <- mahalanobis(alpha.ij*data,rep(0,p),covariance)
#if (!preliminary.mean.imputation) # This condition waived to be compatible with D4
-5.2.1-2.C
{
    MD <- p^2*MD/apply(alpha.ij,1,sum)^2
    if (min(MD)<0) cat("Warning: Negative Mahalanobis distances\n")
}
outind <- 1-(MD > (c*qchisq(2*pnorm(1)-1,p)))
cat("New set of",sum(outind),"outliers generated\n")
}
### List of observations to be imputed ###
 observations.with.errors <- apply(missing.errors,1,sum)<p
to.be.imputed <- which(outind|observations.with.errors)
imputed <- rep(0,n)
### Start of imputation process ###
### List of complete and correct donors ###
 complete.donors <- apply((1-outind)*missing.errors,1,sum)==p
cat("\nNumber of complete and error-free observations: ",sum(complete.donors),"\n")
for (observation in to.be.imputed)
{
    ### Indicator of potential donors for the observation ###
    if (outind[observation]==0 & beta<1)
    {
        potential.donors <- apply((1-outind)*sweep(missing.errors,2,(missing.errors[observation]),"*") ,1,sum)==beta*p &
        apply(sweep(1-missing.errors,2,(1-missing.matrix[observation]),"*") ,1,sum)==0
    } else
    {
        potential.donors <- complete.donors
    }
    potential.donors[observation] <- FALSE # Exclude the observation as its own donor
    switch(sum(potential.donors)+1 ,
    {# 1
        cat("\nNo donor for observation ",observation,"\n")
        cat("All complete error-free observations used as donors.\n")
        potential.donors<-complete.donors
    },
    {# 2
        cat("\nOnly one donor for observation ",observation,"\n")
    })
    if (verbose) cat("Number of potential donors ",sum(potential.donors),"\n")
### Distances of the observation to potential donors ###
 distances.to.donors <- mahalanobis(sweep(data[potential.donors,],2,data[observation,],"-")*
 sweep(alpha.ij[potential.donors,],2, alpha.ij[observation,"*"] ,rep(0,p),
covariance)
    #if (!preliminary.mean.imputation) # This condition waived to be compatible with D4
-5.2.1-2.C
    distances.to.donors <- p^2*distances.to.donors/
        apply(sweep(alpha.ij[potential.donors,],2,alpha.ij[observation,"*"] ,1,sum)^2

### Selection of the donor ###
 min.dist.to.donors <- min(distances.to.donors)
donors are the indices for a vector with the indices of the potential donors

AMELI-WP4-D4.1
# potential donors is an indicator vector of length n

```r
# if (is.na(min.dist.to.donors)) donors<-1:sum(potential.donors) else
if (min.dist.to.donors<0) stop("Warning: Minimal distance to nearest neighbour negative\n") else
donors <- which(distances.to.donors==min.dist.to.donors)
if (length(donors)==1)
{
    imputed[observation] <- which(potential.donors)[donors]}
else{
    imputed[observation] <- which(potential.donors)[sample(donors,1)]
}
if (verbose) {
    cat("Observation",observation,"imputed by donor ",imputed[observation],"\n")
cat("distance to donor: ",min.dist.to.donors,"\n")
}

# *** Imputation ***

new.data <- old.data
new.data[as.logical(outind),] <- old.data[imputed[as.logical(outind)],]
non.outliers.errors <- which((1-outind) & observations.with.errors)
new.data[non.outliers.errors,][missing.errors[non.outliers.errors,]<1] <-
    old.data[imputed/non.outliers.errors,][missing.errors[non.outliers.errors ,]<1]
new.center <- apply(weights*new.data,2,sum)/sum(weights)
new.variances <- apply(weights*sweep(new.data,2,new.center,"-")^2,2,sum)/sum(weights)

# ############ Computation time stop ############

calc.time <- proc.time() - calc.time

# ****************** Results *******************

POEM.r <- list(
    preliminary.mean.imputation=preliminary.mean.imputation,
    size.of.data=dim(data),
    completely.missing=sum(comp.miss),
    sum.of.weights=sum(weights),
    good.values=good.values,
    response.per.variable=apply(missing.matrix,2,sum),
    number.of.nonoutliers.before.reweighting=old.number.of.nonoutliers,
    weighted.number.of.nonoutliers.before.reweighting=old.weighted.sum.of.nonoutliers,
    number.of.nonoutliers.after.reweighting=old.number.of.nonoutliers,
    weighted.number.of.nonoutliers.after.reweighting=old.weighted.sum.of.nonoutliers,
    old.center=new.center, old.variances=new.variances, 
    new.center=new.center, new.variances=new.variances, 
    covariance=covariance, 
    imputed.observations=to.be.imputed, 
    donors=imputed[to.be.imputed], 
    outind=outind)
POEM.i <- new.data

# ****************** Output ******************

cat("\n","POEM has imputed",length(to.be.imputed),"observations(s) in",calc.time,"\nseconds.","\n")
cat("The results are in POEM.r and POEM.i \n")
```

TRC <- function(data, weight, overlap = 3, mincor = 0, 
    robust.regression = "rank", gamma = 0.5, prob.quantile = 0.75, alpha = 0.05, 
    type = "m", output = F) {
# Multivariate Outlier Detection in Survey Data
# TRC algorithm as described in:
# BEGUIN, C. and HULLIGER B., (2002),
Chapter 9. MODI Package

# EUREDIT Workpackage x.2 D4-5.2.1-2.C
# Develop and evaluate new methods for statistical outlier
detection and outlier robust multivariate imputation,
# Program by CEDRIC BEGUIN
# Modification : 27 March 2003 (Beat Hulliger) (R-Version of TRC030313.ssc)
# Modification : 22 December 2005 (Beat Hulliger) Correct standardization of rank
# correlations if no imputation
# Modification : 16 February 2006 (Beat Hulliger) Correct ordering if p=2
# Modification : 30 March 2006 (Beat Hulliger) default md.type="m", default robust.
# regression="rank"
# Note; The data is not standardised to unit scales.
# Copyright : Swiss Federal Statistical Office, 2002
# gamma : minimal overlap of variables for regression
# robust.regression : type of regression "irls" or based on rank correlation
# alpha : Quantile for F-distribution for cut-off
# md.type="c" conditional, md.type="m" marginal Mahalanobis-distance

# Computation time start ############
calc.time <- proc.time()

# Preprocessing ############
if(!is.matrix(data)) data<-as.matrix(data)
n <- nrow(data)
p <- ncol(data)
if (alpha<0.5) alpha<-1-alpha
if (missing(weight)) (weight <- rep(1,n))
if (length(weight)!=n) stop("n Sampling weights of wrong length")
new.indices <- which(apply(is.na(data),1,prod)==0)
discarded<-NA
nfull<-n
if (length(new.indices)<n)
{
discarded<-which(apply(is.na(data),1,prod)==1)
cat("Warning: missing observations",discarded,"removed from the data\n")
weight <- weight[new.indices]
n <- nrow(data)
}
missing.matrix <- 1-is.na(data)
cat("\nNumber of missing items: ", sum(is.na(data)), ", percentage of missing items: ", mean(is.na(data)),"\n")
if (output) cat("End of preprocessing in",proc.time()-calc.time,"seconds \n")

# Estimation of location and scatter ############
if (output) spearman.time <- proc.time()
medians <- apply(data, 2, weighted.quantile, w = weight)
correction.constant.mad <- 1/qnorm(0.75)
mads <- correction.constant.mad*apply(abs(sweep(data, 2, medians, "-")), 2, weighted.quantile, w = weight)
if(sum(mads == 0) > 0)
{
cat("Some mads are 0. Using", prob.quantile, "quantile absolute deviations!\n")
mads <- (1/qnorm(0.5*(1+prob.quantile)))*apply(abs(sweep(data, 2, medians, "-")), 2, weighted.quantile, w = weight, prob = prob.quantile)
if(sum(mads == 0) > 0)
{
cat("The following variable(s) have", prob.quantile, "quantile absolute deviations
equal to 0 ",which(mads == 0),"\n")
stop("Remove these variables or increase the quantile probability\")
}
}
if (prod(missing.matrix)==1) # No missing values
{
weighted.ranks <- matrix(0,n,p)
for (i in 1:p)
{

AMELI-WP4-D4.1
weighted.ranks.[i] <- (apply(data[,i,drop=F],1,.sum.weights,weights=weight,observations=data[,i])
+0.5*apply(data[,i,drop=F],1,.sum.weights,weights=weight,observations=data[,i],lt=F)
+0.5)
}
scatter <- (12*(t(weighted.ranks)[,i]*(weight*weighted.ranks)) / (t(missing.matrix)[,i]*(weight*missing.matrix))^(3-3)
scatter[scatter>1] <- 1
scatter[scatter<-(-1)] <- -1
scatter <- 2*sin(pi*scatter/6)
if (output) {
cat("Spearman Rank Correlations (truncated and standardized):\n")
print(scatter)
cat("End of Spearman rank correlations estimations in","proc.time()=spearman.time,"
seconds\n")
}

if (output) {cat("No imputation\n")
}
else {

scatter <- matrix(0,p,p)
size.of.cor.sets <- t(missing.matrix)%*%missing.matrix
if (sum(size.of.cor.sets<overlap)>0) {
cat("Warning: ",sum(size.of.cor.sets<overlap)/2," couples of variables have less
than ",overlap,
" observations in common, therefore their rank correlations will be set to 0.\n")
}
if (output) cat("Computing Spearman Rank Correlations :\n")
for (i in 1:(p-1)) {
if (output) cat("i=",i,"\n")
for (j in (i+1):p) {
if (output) cat(" j=",j,"\n")
if (size.of.cor.sets[i,j]>=overlap) {

common.observations <- missing.matrix[i]*missing.matrix[j]
weighted.ranks.i <- (apply(data[common.observations],i,drop=F),1,.sum.weights,
weights=weight[common.observations],observations=data[common.observations,i]))
+0.5*apply(data[common.observations],i,drop=F),1,.sum.weights,
weights=weight[common.observations],observations=data[common.
observations,i],lt=F)
+0.5)
weighted.ranks.j <- (apply(data[common.observations],j,drop=F),1,.sum.weights,
weights=weight[common.observations],observations=data[common.
observations,j]))
+0.5*apply(data[common.observations],j,drop=F),1,.sum.weights,
weights=weight[common.observations],observations=data[common.
observations,j],lt=F)
+0.5)
scatter[i,j] <- 12*sum(weight[common.observations]*weighted.ranks.i*weighted.
ranks.j)/sum(weight[common.observations])^(3-3)
}
}
scatter <- scatter+t(scatter)+diag(p)
scatter[scatter>1] <- 1
scatter[scatter<-(-1)] <- -1
scatter <- 2*sin(pi*scatter/6) # Standardization put before imputation 13.03.03 Beat Hulliger
if (output) {

cat("Spearman Rank Correlations (truncated and standardized):\n")
print(scatter)
Chapter 9. MODI Package

### Ad hoc imputation of missing values

```r
imputation.time <- proc.time()
variables.to.be.imputed <- which(apply(missing.matrix, 2, prod)==0)
regressors.cor <- (scatter-diag(p))[variables.to.be.imputed,][size.of.cor.sets[variables.to.be.imputed,]]*(gamma*gamma)
regressors.cor <- as.matrix(t(regressors.cor))
# print(regressors.cor)
if (length(variables.to.be.imputed)>1) regressors.list.ordered <- apply(-abs(regressors.cor),1,order)
for (v in 1:length(variables.to.be.imputed))
{ observations.to.be.imputed <- (missing.matrix[,variables.to.be.imputed[v]])
  r <- 0
  repeat
  { r <- r+1
    if (abs(regressors.cor[v,regressors.list.ordered[r,v]])<mincor)
      { cat("No eligible regressor found for variable",v,"observation(s)",which(observations.to.be.imputed),"\nTry to relax the regressor eligibility conditions.\n")
        stop()
      }
    if (sum(observations.to.be.imputed&missing.matrix[,regressors.list.ordered[r,v]])==0) next
    observations.imputed.by.r.on.v <- which(observations.to.be.imputed&missing.matrix[,regressors.list.ordered[r,v]])
    k <- length(observations.imputed.by.r.on.v)
    common.observations <- missing.matrix[,variables.to.be.imputed[v]]&missing.matrix[,regressors.list.ordered[r,v]]
    if (robust.regression="irls") {
      regression.coef <- rlm(data[common.observations,variables.to.be.imputed[v]]~data[common.observations,regressors.list.ordered[r,v]],weights=weight[common.observations])$coefficients
    } else {
      regression.coef <- c(0,regressors.cor[v,regressors.list.ordered[r,v]]*mads[variables.to.be.imputed[v]]/mads[regressors.list.ordered[r,v]])
    }
data[observations.imputed.by.r.on.v,variables.to.be.imputed[v]] <- matrix(c(rep(1,k),data[observations.imputed.by.r.on.v,regressors.list.ordered[r,v]]),k,2) %*% regression.coef
    if (output) cat("\nVariable",variables.to.be.imputed[v],":\n")
  }
  if (output) cat("End of imputation in","proc.time()-imputation.time","seconds\n")
}
# scatter <- 2*sin(pi*scatter/6) # standardization moved in front of imputation
scatter <- t(t(mads * scatter) * mads)
new.basis <- eigen(scatter)$vectors
data <- data %*% new.basis
center <- apply(data, 2, weighted.quantile, w = weight)
scatter <- (correction.constant.mad*apply(abs(sweep(data, 2, center, ",=")), 2, weighted.quantile, w = weight))^2
if(sum(scatter == 0) > 0)
```

AMELI-WP4-D4.1
cat("Some mads are 0. Using prob.quantile, "quantile absolute deviations\n")
scatter <- ((1/qnorm(0.5*(1+prob.quantile)))^2)*apply(abs(sweep(data, 2, center, "+")),
2, weighted.quantile, w = weight, prob = prob.quantile)^2
if(sum(scatter == 0) > 0)
{
  stop("Please, increase the quantile probability\n")
}
center <- as.vector(new.basis \%*% center)
scatter <- as.matrix(new.basis \%*% diag(scatter) \%*% t(new.basis))
data <- data \%*% t(new.basis)

# Mahalanobis distances
# dist.with.imputed.values <- mahalanobis(data, center, scatter)
# var.with.imputed.values <- var(data)
data[missing.matrix] <- NA
s.patterns <- apply(matrix(as.integer(is.na(data)),n,p),1,paste,sep="",collapse="")
perm <- order(s.patterns)
data <- data[perm,]
s.patterns <- s.patterns[perm]
s.counts <- as.vector(table(s.patterns))
s.id <- cumsum(s.counts)
S <- length(s.id)
missing.items <- is.na(data[s.id,,drop=F])
nb.missing.items <- apply(missing.items,1,sum)
indices <- (!missing.items[1,])
if (md.type=="c") metric <- solve(scatter) else metric <- scatter
dist <- mahalanobis(data[(s.id[1]+1):s.id[indices],indices],
                   t(indices),metric[indices,drop=F],inverted=(md.type=="c"))*p/(p-nb.missing.items[1])
if (S>1)
{
  for (i in 2:S)
  {
    indices <- (!missing.items[1,])
    dist <- c(dist,mahalanobis(data[(s.id[i-1]+1):s.id[i],indices,drop=F],center[indices],
                               metric[indices,indices,drop=F],inverted=(md.type=="c"))*p/(p-nb.missing.items[1]))
  }
}

# Choice of outliers
# Nominate the outliers using the original numbering (without discarded obs.)
cutpoint<-qf(alpha,p,n-p)/qf(0.5,p,n-p)*median(dist)
dist<dist[order(perm)]
good <- (1:n)[dist < cutpoint]
outliers <- (1:n)[-good]

# outliers and distances in original numbering with full dataset
outn<-logical(n)
outn[outliers]<-TRUE
outnfull<-logical(nfull)
outnfull[new.indices]<-outn
distnfull[new.indices]<-dist

calc.time <- proc.time() - calc.time

# Results
TRC.r <- list(sample.size = n,
              number.of.variables = p,
              number.of.missing.items=nb.missing.items,
              significance.level = alpha,
              computation.time = calc.time,
              medians=medians, mads=mads,
              center = center,
scatter = scatter,
robust.regression=robust.regression,
md.type=md.type,
cutpoint=cutpoint)
TRC.i <- list(outind = outnfull,
dist = distnfull)

############ Output ############

cat("\n", "TRC has detected", length(outliers), "outlier(s) in", calc.time[1], "seconds \n\n")
cat(" The results are in TRC.r and TRC.i","\n")

EAimp <-
function(data, weights , outind=EAdet.i$outind,
duration=EAdet.r$duration,
maxl = 5, kdon=1, monitor = FALSE, threshold=FALSE, deterministic=TRUE,
fixedprop=0 )
{
# EPIDEMIC Algorithm for Multivariate Outlier Detection
# Program by CEDRIC BEGUIN and BEAT HULLIGER
# Created : Wednesday, January 24, 2001
# Last modification : 4 August 2009 Beat Hulliger
# Conversion to R from EA030313.ssc by Beat Hulliger (4.7.2003)
# Modular programming and packaging: Beat Hulliger 27.3.2009
# discrete : if TRUE changes the correction for missing values (instead of \( \frac{(p-q)}{p}\) when summing (corresponds to corr=0.5)
# reach: Transmission.distance. reach=0 yields maximal distance. Default is \( \frac{1}{p+1}\) n
quantile of minimal distance to nearest neighbor
# usable.only : Sets observations with more than half the variables missing to all missing (never infected)
# transmission.function: "step", "linear", "power" or "root"
# tf. const: constant for transmission functions power (default=p) and root (default=maxl)
# distance type: The types for function dist()
# maxl: Maximum number of steps without change (finish)
# prob.quantile: If mads fail then take this quantile absolute deviation of abs. deviations
# threshold: Infect all points with probability above \( 1-0.5^{-\frac{1}{maxl}}\)
# fixedprop: Fixed proportion to be infected
# deterministic: Infect points with largest prob. (expected number)
# outind: a logical vector with TRUE if an outlier.
# EAdet must have been run before in order to calculate the distances and counterprobabilities.
# EAdet stores the counterprobabilities in a global vector EA.distances and some parameters in EA.distances.parameters. EAimp uses these counterprobabilities and parameters.
# No changes to the transmission function are possible in EAimp.
#
# Computation time start ############

calc.time <- proc.time()[1]

### Dimensions ###

# use all data: complete non-responses are to be replaced completely (similar to outliers )

n <- nrow(data)
p <- ncol(data)
if (missing(weights)) weights<-rep(1,n)
response<-=!is.na(data)
complete.records <- apply(response, 1, prod)
usable.records <- apply(response, 1, sum) >= p/2
cat("\nNumber of complete records ", sum(complete.records))
cat("\nNumber of records with maximum p/2 variables missing ", sum(usable.records))

Standardization of weights to sum to sample size
np <- sum(weights)
weights <- as.single((n * weights)/np)

Distances, i.e. counterprobabilities, must have been stored in a global variable EA.dist.
dist.par <- EA.dist.parameters
cat("\n Global variable EA.distances used")
if (length(EA.distances)!=(n*(n-1))/2) cat("\n Distances not on same number of observations")

Imputation with reverse epidemic and random imputation among nearest potential donors

Preparation
imp.data <- data
to.impute <- apply(response, 1, prod) ==0 | outind
imp.ind <- which(to.impute)
n.imp <- length(imp.ind)
cat("\n Number of imputands is ", n.imp)

# reach d0 is set to maximum in order to reach all outliers (max.min.di of EA.dist)
d0 <- EA.dist.par[2]
cat("\n Reach for imputation is ", d0)

Start of imputations
for (i in 1:n.imp) {
# Start epidemic at the observation that needs imputation
imputand <- imp.ind[i]
imp.time <- 1
imp.infected <- rep(FALSE, n)
imp.infected[imputand] <- TRUE

if (outind[imputand]) missvar <- which(!response[imputand,])
else missvar <- which( !response[imputand,])

# donors must have values for the missing ones and must not be an outlier
potential.donors <- apply(as.matrix(response[,missvar]), 1, sum) == n.missvar & !outind
potential.donors[imputand] <- FALSE

if (monitor) {
cat("\n Imputand: ", imputand)
cat(" missvar: ", missvar)
cat(" # pot. donors: ", sum(potential.donors))
}

# if no donor can impute all missing values of the imputand then maximise number of
# imputed values
if (sum(potential.donors)==0) {
  max.miss.val <- max(apply(as.matrix(response[-imputand,missvar]), 1, sum))
cat("\n No common donor for all missing values. ", n.missvar - max.miss.val, " missing
values will remain")
potential.donors <- apply(response[,missvar], 1, sum) == max.miss.val
}

# donors must have values for the missing ones and must not be an outlier
potential.donors <- apply(as.matrix(response[,missvar]), 1, sum) == max.miss.val

# donors must have values for the missing ones and must not be an outlier
donors <- rep(FALSE, n)
new.imp.infected <- imp.infected
imp.infected <- sum(imp.infected)

hprod <- rep(1, n)
imp.infection.time <- rep(0, n)
imp.infection.time[imp.infected] <- imp.time

# Run epidemic from imputand until at least one potential donor is infected
repeat {
  if (sum(donors) >= kdon | imp.time > duration) break
  #cat("\n imp.time = ", imp.time, ", imp.infected = ", n.imp.infected)
  imp.time <- imp.time + 1
  imp.infected <- imp.infected + hprod

  if (imp.infection.time[imp.infected] == duration)
    imp.infection.time[imp.infected] <- imp.time
  else
    imp.infection.time[imp.infected] <- imp.infection.time[imp.infected] + 1
}

© http://ameli.surveystatistics.net/ - 2011
```r
# print(memory.size())
imp.time <- imp.time * 1
old.imp.infected <- imp.infected
if (sum(new.imp.infected) > 1) {
  hprod[!imp.infected] <- hprod[!imp.infected] * apply(sweep(sweep(matrix(EA.distances[apply(as.matrix(which(!imp.infected)),
                                            1, .ind.dijis, js = which(new.imp.infected), n = n)],
                                           sum(new.imp.infected), n = n
                                           .imp.infected),1,
                                           weights[new.imp.infected],"-";2,weights[!imp.infected],"-"), 2, prod)
} else {
  if (sum(new.imp.infected) == 1)
    hprod[!imp.infected] <- hprod[!imp.infected] * EA.distances[apply(as.matrix(
      which(!imp.infected)), 1, .ind.dijis, js = which(new.imp.infected), n = n)
    ^ weights[new.imp.infected], weights[!imp.infected]) ^ prod]
  else
    imp.infected[!imp.infected] <- rank(1 - hprod[!imp.infected], ties = "max") >= n - n.imp.infected
}
if (deterministic) {
  n.to.infect <- max(1, round(sum(1 - hprod[!imp.infected]))) # At least 1 infection at each step
  # Rank is maximum to allow infection
  imp.infected[!imp.infected] <- rank(1 - hprod[!imp.infected], ties = "max") >= n - n.imp.infected
  n.imp.infected <- sum(imp.infected)
  imp.infection.time[new.imp.infected] <- imp.time
  donors <- potential.donors & imp.infected
  if (monitor) cat(" donors: ", which(donors))
  next
} else {
  if (imp.time > duration) cat("No donor found")
  else {
    if (sum(donors) > 1) don.imp <- sample(which(donors), 1)
    else don.imp <- which(donors)
    if (monitor) cat(" # donors: ", sum(donors), ", chosen donor: ", don.imp)
    imp.data[imputand, missvar] <- data[don.imp, missvar]
  }
}
# cat("n memory use",memory.size())
}
calc.time <- round(proc.time()[1] - calc.time, 5)
cat("n remaining missing values is ", sum(is.na(imp.data)))

# Results
EAimp.r <- list(sample.size = n, number.of.variables = p, n.complete.records = sum(complete.records), n.usable.records = sum(usable.records), duration = duration, reach = d0, threshold = threshold, deterministic = deterministic, computation.time = calc.time)
EAimp.data <- imp.data

# Output
EAimp.data <- function(data, weights, reach = "max", transmission.function = "root", power = ncol(data), distance.type = "euclidean",
```
global.distances=F, maxl = 5, plotting = T, monitor = F, prob.quantile = 0.9, random.start = F, fix.start, threshold=F, deterministic=TRUE, remove.missobs=FALSE)
{
# EPIDEMIC Algorithm for Multivariate Outlier Detection
#
data:
# the epidemic algorithm and transformed rank correlations, JRSS-A, 167, Part 2, pp.
# 275-294.
#
# Program by CEDRIC BEGUIN and BEAT HULLIGER
# Created : Wednesday, January 24, 2001
# Last modification : 4 August 2009 Beat Hulliger
# Conversion to R from EA030313.ssc by Beat Hulliger (4.7.2003)
# Modular programming and packaging : Beat Hulliger 27.3.2009
#

if (!is.matrix(data)) data<-as.matrix(data)
n <- nrow(data)
p <- ncol(data)
if (missing(weights)) weights <- rep(1,n)
# Removing the unit(s) with all items missing
new.indices <- which(apply(is.na(data),1,prod)==0)
discarded<-NA
nfull<-n
if ((length(new.indices)<n) & remove.missobs)
{
discarded<-which(apply(is.na(data),1,prod)==1)
cat("Warning: missing observations",discarded,"removed from the data\n")
data <- data[new.indices]
weights <- weights[new.indices]
n <- nrow(data)
}
complete.records <- apply(!is.na(data),1,prod)
usable.records <- apply(!is.na(data),1,sum) >= p/2
cat("\nDimensions (n,p):", n, p)
cat("\nNumber of complete records ", sum(complete.records))
cat("\nNumber of records with maximum p/2 variables missing ", sum(usable.records),"\n")
power <- as.single(power)
# Standardization of weights
np <- sum(weights)
weights <- as.single((n + weights)/np) #
# Computation time start #######
calc.time <- proc.time()[1] #
# Calibration and setup #######
medians <- apply(data, 2, weighted.quantile, w = weights, prob = 0.5)
data <- sweep(data, 2, medians, "-")
mads <- apply(abs(data), 2, weighted.quantile, w = weights, prob = 0.5)
qads <- apply(abs(data), 2, weighted.quantile, w = weights, prob = prob.quantile)
if(sum(mads == 0) > 0) {
cat("\nSome mads are 0. Standardizing with ", prob.quantile, " quantile absolute
deviations!"")
}
if(sum(qads == 0) > 0)
cat("\nSome quantile absolute deviations are 0. No standardization!"")
else data <- sweep(data, 2, qads, "/")
else data <- sweep(data, 2, mads, "/")
if(monitor) {
standardized.data <- data
cat("\nmemory use after standardisation: ",memory.size())}
# # Calculation of distances
if (global.distances==T) {
  EA.dist.par<-EA.distances.parameters
cat("\n\n Global variable EA.distances used")
} else {
  EA.dist.par<-EA.dist(data, n,n,p=p,weights = weights,reach=reach,
    transmission.function = transmission.function, power=power, distance.type =
    distance.type, 
  maxl = maxl, monitor = monitor,calc.time=calc.time)
cat("\n\n Distances finished")
}

# The distances calculated by EA.dist are the counterprobabilities in single precision.
# These counterprobabilities are stored as a global variable EA.distances to avoid
# copying this very large vector.
if (monitor)
cat("\n memory use after distances : ",memory.size())
cat("\n Index of sample spatial median is ",EA.dist.par[1])
cat("\n Maximal distance to nearest neighbor is ", EA.dist.par[2])
cat("\n Transmission distance is ", EA.dist.par[3] , ",\n")

# ########### Initialisation ############
#
cat("\n
 Initialisation of epidemic ")
comp.time.init <- proc.time()[1] - calc.time
if (monitor)
cat("\n Initialisation time is ", comp.time.init)
if (random.start)
  start.point <- sample(1:n, 1, prob = weights)
else {
  if (! missing(fix.start))
    start.point <- fix.start
  else start.point <- EA.dist.par[1]
}
time <- 1
infected <- rep(F, n)
infected[start.point] <- T
new.infected <- infected
n.infected <- sum(infected)
hprod <- rep(1, n)

infection.time <- rep(0, n)
infection.time[start.point] <- time
# # # # # # # Main loop # # # # # # # #
#
repeat {
  if (monitor) cat("\n time = ", time , ", infected = ", n.infected)
  #print(memory.size())
time <- time + 1
old.infected <- infected
if (sum(new.infected) > 1) {
  hprod[!infected] <- hprod[!infected] * apply(sweep(sweep(matrix(EA.distances[apply(as.matrix(which(!infected)),
    1,.ind.dij, js = which(new.infected), n = n)], sum(new.infected), n - n.
    infected),1,
    weights[new.infected],"^" ,2,weights[!infected],"^")),2,prod)
} else {
  if (sum(new.infected) == 1)
    hprod[!infected] <- hprod[!infected] * EA.distances[apply(as.matrix(which(!
    infected)), 1,.ind.dij, js = which(new.infected), n = n)]*(weights[new.infected]*
    weights[!infected])
  }
  if (deterministic) {
    n.to.infect <- sum(1 - hprod[!infected]) # HRK: expected number of infections
    # Do maxl trials for very small inf. prob.
    if (n.to.infect<0.5) n.to.infect <- sum(1 - hprod[!infected]-maxl)
    n.to.infect <= round(n.to.infect)
    infected[!infected] <- round(1 - hprod[!infected])>n-n.infected+n.to.infect
  } else {
  if (threshold) {infected[!infected] <- hprod[!infected] <=0.5*(1/maxl) else
  AMELI-WP4-D4.1
infected[n.infected] <- as.logical(rbinom(n - n.infected, 1, 1 - hprod[n.infected])
new.infected <- infected & (!old.infected)
n.infected <- sum(infected)
infection.time[new.infected] <- time
if(n.infected == n) { break
} if((time - max(infection.time)) > maxl) { break
}
next
duration <- max(infection.time)
if(monitor) { last.infection.prob <- 1 - hprod
cat("\n memory use after epidemic: ",memory.size())
}
#
# Impute infection.time for not infected #############
# This is to show better the healthy on a graph of infection times
#
infection.time[!infected] <- ceiling(1.2 * duration) #
# Computation time stop #############
#
calc.time <- round(proc.time()[1] - calc.time, 5)
med.infection.time <- weighted.quantile(infection.time,weights,0.5)
mad.infection.time <- weighted.quantile(abs(infection.time-med.infection.time),weights ,0.5)
cat("\n med and mad of infection times: ",med.infection.time, " and ",mad.infection.time
.dom (mad.infection.time=0) mad.infection.time <- med.infection.time
if (mad.infection.time==0) mad.infection.time < med.infection.time
cutpoint <- min(med.infection.time+3*mad.infection.time,duration)
cat("\\n Proposed cutpoint is ",min(cutpoint,duration))
 outlier.ind <- as.numeric(rownames(data))[which(infection.time>=cutpoint)]
# Blowing up to full length
infectedn<-logical(n)
infection[infected]<- TRUE
infectedfinal[logical(nfull)]<-infectedn else infectedfinal[<infectedn
time<-rep(NA,nfull)
infection[!new.indices]<infection.time else time<-infection.time
time[!new.indices]<time=cutpoint
outlier<time=cutpoint
#
# Results #############
#
EAdet.r <- list(sample.size = n, discarded=discarded, discarded, number.of.variables = p, n.complete.records = sum(complete.records), n.usable.records = sum(usable.records), medians = medians, mads = mads, prob.quantile
= prob.quantile, quartile.deviations = qads, start = start.point, transmission = transmission.function = transmission
.function, power=power, maxl=maxl,
min.nn.dist=E.dist.par[2], transmission.distance = E.dist.par[3], threshold=threshold, distance.type = distance.type,
deterministic=deterministic, number.infected = n.infected,
cutpoint=cutpoint, number.outliers=sum(outlier), outliers=outlier.ind,
duration = duration, computation.time = calc.time, initialisation.computation.time = comp.time.init)
EAdet.i <- list(infected = infectedfull, time = time, outind=outlier)
# plotting
if(plotting) { ord <- order(infection.time)
plot(infection.time[ord],cumsum(weights[ord]), ylab = "cdf of infection time")
abline(v=cutpoint) #
}
#
# Output #############
#
cat("\n EA detection has finished with", n.infected, "infected points in", calc.time ", seconds.")
cat("\n The results are in EAdata.r and EAdata.i", "\n")
Chapter 9. MODI Package

```r
EA.dist <-
function(data, n,p,weights,reach,transmission.function, power, distance.type, maxl, monitor,calc.time)
{
# Calculation of distances for EPIDEMIC Algorithm for multivariate outlier detection and imputation
# This is a utility for EAD and EAI
# Modular programming by Beat Hulliger
#
# save distances, i.e. the latter counterprobabilities, as global vector
EA.distances <- as.single(dist(data, method = distance.type))
#
The dist function handles missing values correctly except if there is no overlap (see counterprob)
if(monitor) cat("", Computation time is ", proc.time()[1] - calc.time) 
min.di <- rep(0, n) #
means.di <- rep(0, n) # Will be used for the sample spatial median and for d0
for(i in 1:n) {
  di <- EA.distances[ind.dij(i, 1:n, n)]
  min.di[i] <- nz.min(di) # weighted mean of distances to account for missing distances
  means.di[i] <- sum(di * weights[ - i ], na.rm = T)/sum(weights[ - i][!is.na(di)])
}
# Save min.di and means.di for later inspection
if(monitor) { 
  min.di <<- min.di
  means.di <<- means.di
}
#
# Sample spatial median
# Restrict to usable observations (for sample spatial median)
usable.records <- apply(!is.na(data), 1, sum) >= p/2
means.di.complete <- means.di
means.di.complete[usable.records] <- NA
sample.spatial.median.index <- which(means.di.complete == min(means.di.complete, na.rm = T))[1] # Determine tuning distance d0
max.min.di <- max(min.di, na.rm = T)
d0 <- switch(EXPR = as.character(reach),
  max=min(max.min.di,2*sqrt(p)),
  quant=min(weighted.quantile(min.di, w = weights, prob = 1-(p+1)/n),
  2 * sqrt(p)),
  reach)
if(reach=="max") {d0 <- min(max.min.di,2*sqrt(p))} else 
{d0 <- min(weighted.quantile(min.di, w = weights, prob = 1-(p+1)/n), 2 * sqrt(p))}
#
# Calculation of counterprobabilities
# counterprobabilities stocked in distances vector to save memory
# counterprobabilities are set to 1 if missing
if (n%%2==0) {
  l.batch <- n-1
  n.loops <- n/2
} else {
  l.batch <- n
  n.loops <- (n-1)/2
} 
if(transmission.function == "step") {
  for(i in 1:n.loops) {
    dij <- EA.distances[(i-1)*l.batch+(1:l.batch)]
    dij <- as.single(dij > d0)
    dij[is.na(dij)] <- 1
    EA.distances[(i-1)*l.batch+(1:l.batch)] <<- as.single(dij)
  }
} else {


AMELI-WP4-D4.1
```

if(transmission.function == "linear") {
    for(i in 1:n.loops) {
        dij <- EA.distances[(i-1)*l.batch+(1:l.batch)]
        dij <- 1 - max(0, d0 - dij)/d0
        dij[is.na(dij)] <- 1
        EA.distances[[(i-1)*l.batch+(1:l.batch)] <- as.single(dij)
    }
} else {
    if(transmission.function == "power") {
        beta <- as.single((0.01^(-1/power) - 1)/d0)
        for(i in 1:n.loops) {
            dij <- EA.distances[(i-1)*l.batch+(1:l.batch)]
            dij <- 1 - (beta * dij + 1)^(-power)
            dij[is.na(dij)] <- 1
            EA.distances[[(i-1)*l.batch+(1:l.batch)] <- as.single(dij)
        }
    } else { # default transmission function is the root function
        for(i in 1:n.loops) {
            dij <- EA.distances[(i-1)*l.batch+(1:l.batch)]
            dij <- 1 - (1 - dij/d0)^(1/power)
            dij[is.na(dij)] <- 1
            EA.distances[[(i-1)*l.batch+(1:l.batch)] <- as.single(dij)
        }
    }
} # main result is in global variable EA.distances
# save minimal distances to nearest neighbour for inspection
EA.min.di <- min.di
# save some additional derived parameters in a global vector
EA.distances.parameters <- c(sample.spatial.median.index, max.min.di, d0)

EM.normal <- function(data, weights=rep(1,nrow(data)), n=sum(weights), p=ncol(data), s.counts, s.id, S, T.obs, start.mean=rep(0,p), start.var=diag(1,p), numb.it=10, Estep.output=F) {
    # This version of EM does not contain the computation of the observed sufficient statistics,
    # they will be computed in the main program of BEM and passed as parameters as well as the
    # statistics on the missingness patterns.
    #
    # Initialization
    # Creates theta which is the matrix form of the initial parameter used by EM
    theta <- matrix(0,p+1,p+1)
    theta[1,1] <- -1
    theta[1,2:(p+1)] <- theta[2:(p+1),1] <- start.mean
    theta[2:(p+1),2:(p+1)] <- start.var
    # Iterations of EM
    for (boucle in 1: numb.it) {
        if (Estep.output) cat("E-step ",boucle,"\n")
        # The E-step
        # Initializing T.tot to T.obs
}
# ER Algorithm for Multivariate Outlier Detection in Incomplete Multivariate Normal Data
# ER Algorithm according to Little, R. and P. Smith (1987)
# Implementation based on the BEM algorithm as described in:
# BEGUIN, C. and NULLIGER B., (2002),
# EUREDIT Workpackage x.2 D-5.2.1-2.C
# Develop and evaluate new methods for statistical outlier
detection and outlier robust multivariate imputation,
# ER algorithm is described in: Little, R. and P. Smith (1987).
# Editing and imputation for quantitative survey data.

```r
# T.tot <- T.obs
# Start loop on missing patterns s from 1 to S
# for (s in 1:S)
#   # Identification of the indices of x.mis and x.obs
#   x.mis.id <- (1:p)[is.na(data[s.id[s]])]
#   x.obs.id <- (1:p)[-x.mis.id]
#   # Sweep of theta over the indices of x.obs
#   C.s <- theta
#   for (k in x.obs.id)
#     if (C.s[k+1,k+1]!=0)
#       C.s <- .sweep.operator(C.s,k+1)
#   # Start loop over all observations x having missing pattern s
#   for (i in 1:s.counts[s])
#     if (s==1) {
#       x <- data[i,]
#       weight <- weights[i]
#     } else {
#       x <- data[s.id[s-1]+i,]
#       weight <- weights[s.id[s-1]+i]
#     }
#   # Computation of x.star=E(x.mis|x.obs)
#   x.star <- x
#   for (k in x.mis.id)
#     x.star[k] <- C.s[1,k+1]+sum(C.s[x.obs.id+1,k+1]*x[x.obs.id])
#   # Updating T.tot
#   T.tot[1,] <- T.tot[,1] <- T.tot[1,1]+weight*c(1,x.star)
#   T.tot[2:(p+1),2:(p+1)] <- T.tot[2:(p+1),2:(p+1)]+weight*(x.star%*%t(x.star))
#   T.tot[x.mis.id+1,x.mis.id+1] <- T.tot[x.mis.id+1,x.mis.id+1]+weight*C.s[x.mis.id+1,x.mis.id+1]
# }
# The M-step
# theta <- .sweep.operator(T.tot/n,1)
# End of EM
# return(theta)
```
Main program; will use EM with robustness weights for computation of a location and
scatter estimate when items are missing.

```r
ER <- function(data, weights, alpha = 0.01, psi.par = c(2, 1.25), em.steps = 100, steps.output = F) {
  # Preprocessing of the data

  if (!is.matrix(data)) data <- as.matrix(data)
  n <- nrow(data)
  p <- ncol(data)
  if (alpha < 0.5) alpha <- 1 - alpha
  new.indices <- which(apply(is.na(data), 1, prod) == 0)
  if (length(new.indices) < n) {
    cat("Warning: missing observations ", which(apply(is.na(data), 1, prod) == 1), "removed from the data\n")
    data <- data[new.indices,]
    weights <- weights[new.indices]
    n <- nrow(data)
  }
  if (steps.output) cat("End of preprocessing\n")

  # Computation time start
  calc.time <- proc.time()

  # Order the data by missingness patterns:
  s.patterns <- vector of length n, with the missingness patterns stocked as strings of
  the type "11010...011" with "1" for missing.
  # data, weights and s.patterns ordered using s.patterns' order
  s.patterns <- apply(matrix(as.integer(is.na(data))), n, p), 1, paste, sep="", collapse="")
  perm <- order(s.patterns)
  data <- data[perm,]
  s.patterns <- s.patterns[perm]
  weights <- weights[perm]

  # Missingness patterns stats:
  # s.counts = counts of the different missingness patterns ordered alphabetically.
}
```
# s.id = indices of the last observation of each missingness pattern in the dataset ordered by missingness pattern.
# S = total number of different missingness patterns
# missing.items = missing items for each pattern
# nb.missing.items = number of missing items for each pattern

s.counts <- as.vector(table(s.patterns))
s.id <- cumsum(s.counts)
S <- length(s.id)
missing.items <- is.na(data[s.id,, drop=F])

nb.missing.items <- apply(missing.items,1,sum)

if (steps.output) cat("End of missingness statistics\n")

# ################## Preparation for call to ER #################
#
# This is the old BEM step 1 for the choice of the initial good subset
#
#
# mean and covariance matrix computed by ER
if (nb.missing.items[1]==0)
{
  # Case where some observations are complete (no missing item) =>
  # start.mean and start.var are calculated on the complete observations
  cov.complete <- cov.wt(data[1:s.counts[1],],wt=weights[1:s.counts[1]])/sum(weights[1:s.counts[1]]))
  mean.start <- cov.complete$center
  var.start <- cov.complete$cov
}
else
{
  # Case where all observations have missing items
  mean.start <- apply(data,2,weighted.mean.na,v=weights)
  var.start <- diag(apply(data,2,weighted.var.na,v=weights,na.rm=T))
}

if (steps.output) cat("\n","start.mean: ",mean.start,"\n","start.var: ",var.start)

ER.result <- ER.normal(data=data,weights,psi.par=psi.par,np=sum(weights),p=p,
  s.counts=s.counts,s.id=s.id,S,
  missing.items=missing.items, nb.missing.items ,
  start.mean=mean.start, start.var=var.start,numb.it=em.steps,Estep.
  output=Estep.output)

ER.mean <- ER.result[1,2:(p+1)]
ER.var <- ER.result[2:(p+1),2:(p+1)]

calc.time <- proc.time() - calc.time

# Nominate the outliers using the original numbering

dist <- temp.ER.dist # temp.ER.dist is passed as global variable

good <- dist < qchisq(alpha,p)
outliers <- perm[!good]

# Results

ER.r <- list(sample.size = n,
  number.of.variables = p,
  significance.level = alpha,
  computation.time = calc.time,
good.data = perm[good],
outliers = outliers,
center = ER.mean,
scatter = ER.var,
dist = dist[order(perm)],
robweights = temp.ER.rob.weights

# >>>>>>>>>>>>>>> Output >>>>>>>>>>>>>>>
#
if (!break.flag)
cat("\n","ER has detected\n\n\n","sum(!good),"outlier(s) in"calc.time,"seconds.","\n","\n")
cat("\n","ER did not converge."\n")
cat("\n","\n")
cat("\n",initial.basic.subset.size, final.basic.subset.size,"\n")
cat("\n",number.of.iterations, computation.time, good.data,"\n")
cat("\n",outliers, center, scatter, dist","\n","\n")
}

########################################################
# ER for multivariate normal data
########################################################

ER.normal <- function(data, weights=rep(1,nrow(data)), psi.par=c(2,1.25), np=sum(weights)
 ,p=ncol(data), s.counts, s.id, S, missing.items, nb.missing.items,
 start.mean=rep(0,p), start.var=diag(1,p), numb.it=10, Estep.
 output=F, tolerance=1e-06)
{

 theta <- matrix(0,p+1,p+1)
 theta[1,1] <- -1
 theta[1:(p+1)] <- theta[2:(p+1),1] <- start.mean
 theta[2:(p+1),2:(p+1)] <- start.var
 if (Estep.output) cat("\n","theta: \n",theta)

 break.flag <- F

 rob.weights <- rep(1,nrow(data))
 np.bat <- np

 for (boucle in 1: numb.it)
 {
 if (Estep.output) cat("E-step ",boucle)

 T.tot <- matrix(0,p+1,p+1)
 old.theta <- theta

 for (s in 1:S)
 {
  x.mis.id <- (1:p)[is.na(data[s.id[s]])]
  x.obs.id <- (1:p)[-x.mis.id]

  ...
# Sweep of theta over the indices of x.obs
# C.s <- theta
for (k in x.obs$id)
  if (C.s[k+1,k+1]!=0)
    {C.s <- sweep.operator(C.s,k+1)
    # Start loop over all observations x having missing pattern s
    # for (i in 1:s.counts[s])
      {
        if (s==1)
          {
            x <- data[i,]
            weight <- weights[i]
            rob.weight <- rob.weights[i]
          }
        else
          {
            x <- data[s.id[s-1]+i,]
            weight <- weights[s.id[s-1]+i]
            rob.weight <- rob.weights[s.id[s-1]+i]
          }
        # Computation of x.star=E(x.mis|x.obs)
        x.star <- x
        for (k in x.mis.id)
          x.star[k] <- C.s[1,k+1]+sum(C.s[x.obs.id+1,k+1]*x.obs.id)
        # robustification with robustness weights from last iteration
        x.star <- x.star
        # Updating T.tot
        T.tot[1,] <- T.tot[1,] + rob.weight*weight*C.s[1,1]
        T.tot[2:(p+1),2:(p+1)] <- T.tot[2:(p+1),2:(p+1)] + rob.weight*weight*(x.star%*%t(x.star))
        T.tot[x.mis.id+1,x.mis.id+1] <- T.tot[x.mis.id+1,x.mis.id+1] + rob.weight*weight*C.s[x.mis.id+1,x.mis.id+1]
      }
    # The M-step
    np.hat <- sum(weights*rob.weights)
    theta <- sweep.operator(T.tot/np.hat,1)
    if (Estep.output) cat("\n","theta: \
    ER.mean <- theta[1,2:(p+1)]
    ER.var <- theta[2:(p+1),2:(p+1)]
    indices <- (!missing.items[1,])
    dist <- mahalanobis(data[1:s.id[1],indices,drop=F],ER.mean[indices],ER.var[indices,indices])*p/(p-nb.missing.items[1])
    if (S>1)
      {
        for (i in 2:S)
          {
            indices <- (!missing.items[i,])
            dist <- c(dist,mahalanobis(data[(s.id[i-1]+1):s.id[i],indices,drop=F],ER.mean[indices],ER.var[indices,indices,drop=F])*p/(p-nb.missing.items[i]))
          }
      }
    # new robustness weights
    rob.weights <- ifelse(dist>0,.psi.lismi(sqrt(dist),apply(!is.na(data),1,sum),psi.par=psi.par)/sqrt(dist),1)
    ER.rob.weights <- rob.weights
    if (Estep.output) cat(" Delta: ",signif(max(abs(theta-old.theta)),8),"\n")
    if (max(abs(theta-old.theta))<tolerance) (break.flag<<T;break)
weighted.var <- function(x, w, na.rm = FALSE) {
  if (missing(w))
  w <- rep.int(1, length(x))
  else if (length(w) != length(x))
    stop("'x' and 'w' must have the same length")
  if (is.integer(w))
    w <- as.numeric(w)
  if (na.rm) {
    w <- w[i <- !is.na(x)]
    x <- x[i]
  }
  sum(w*(x-weighted.mean(x,w))^2)/(sum(w)-1)
}

winsimp <- function(data, center, scatter, outind, seed = 1000003)
# Imputation under the multivariate normal model after winsorization
# Beat Hulliger
# 22.5.2009, 7.8.2009
{
  outind <- as.logical(outind)
  # Mahalanobis distance
  data.wins <- as.matrix(data)
  MD <- sqrt(mahalanobis(data.wins, center, scatter))
  cutpoint <- min(MD[outind])
  # robustness weight
  u <- ifelse(MD <= cutpoint, 1, cutpoint/MD)
  # winsorization for outliers
  data.wins[outind,] <- as.matrix(sweep(sweep(sweep(data[outind,],2,center,'-' ),
                            1,u[outind],'*'),
                            2,center,'+
  ))
  # imputation for missing values
  rngseed(seed)
  s <- prelim.norm(data.wins)
  winsimp.r <- imp.norm(s, makeparam.norm(s, list(center, scatter)), data.wins)
  return("Imputed data is in winsimp.r")
}

.ind.dij <- function(i, j, n)
# Addressing function for Epidemic Algorithm
# BEGUIN, C. 2001
{(i - 1) * n - ((i + 1) * i)/2 + j

.ind.dijs <- function(i, js, n)
# Addressing function for Epidemic Algorithm
# BEGUIN, C. 2001
indices <- c(.ind.dij(js[js < i], i, n), .ind.dij(i, js[js > i], n))
return(indices[!is.na(indices)])

.psi.lismi <- function(d, present, psi.par = c(2,1.25))
{
  # psi-function
  # Defined in Little and Smith for ER algorithm
}
a <- psi.par[1]
b <- psi.par[2]
d0 <- sqrt(present)*a/2
return(ifelse(d-d0<=0, d, d0*exp(-((d-d0)^2/(2*b^2)))))

sweep.operator <- function(M,k, reverse = FALSE)
{
  # Definition of the sweep and reverse-sweep operator (Schafer pp 159-160)
  if (reverse) {
    Gjk <- M[,k]
    Hkk <- -1/M[k,k]
    M <- M+(Gjk%*%t(Gjk))*Hkk
    M[k,k] <- Hkk
    return(M)
  }
  else {
    Gjk <- M[,k]
    Hkk <- 1/M[k,k]
    M <- M-(Gjk%*%t(Gjk))*Hkk
    M[k,k] <- -Hkk
    return(M)
  }
}

sum.weights <- function(observations, weights, value, lt = TRUE)
{
  if (lt) return(sum(weights*(observations < value), na.rm = T))
  else return(sum(weights*(observations == value), na.rm = T))
}

.nz.min <- function(x)
{
  nz.min.temp <- min(x[x != 0], na.rm = T)
  if (is.infinite(nz.min.temp)) return(NA) else return(nz.min.temp)
}
Chapter 10

Robust Imputation for Compositions

```r
`aDist` <- function(x, y)
{
  ### computes the Aitchison Distance
  ### needs da.C
  ### Author: Matthias Templ
  ### Licence: GPL 2.0
  if(is.vector(x)) x <- matrix(x, ncol=length(x))
  if(is.vector(y)) x <- matrix(y, ncol=length(y))
  matOrig <- as.numeric(t(x))
  matImp <- as.numeric(t(y))
  n <- dim(x)[1]
  p <- dim(x)[2]
  dims <- as.integer(c(n, p))
  rowDists <- as.numeric(rep(0.0, n))
  distance <- as.numeric(0.0)
  out <- .C("da",
             matOrig,
             matImp,
             dims,
             rowDists,
             distance,
             PACKAGE="robCompositions", NUGX=TRUE
          )[[5]]
  return(out)
}
```

```r
`impCoda` <- function(x, maxit=10, eps=0.5, method="ltsReg", closed=FALSE,
                       init="KNN", k=5, dl=rep(0.05, ncol(x)), noise=0.1)
{
  ## Imputation of compositional data using robust sequential imputation
  ## and the isometric log-ratio transformation
  ## Authors: Matthias Templ and Karel Hron
  ## Licence: GPL 2.0
  ## for method pca: classical, mcd, gridMAD
  ## for regression: lm, ltsReg
  ## if closed == FALSE, ilr is applied.
  if( is.vector(x) ) stop("x must be a matrix or data frame")
  stopifnot((method %in% c("ltsReg", "ltsReg3", "classical", "lm", "roundedZero")))
  if( k > nrow(x)/4 ) warning("k might be too large")
  if(method == "roundedZero") init <- "roundedZero"
  xcheck <- x
  if(method == "roundedZero"){
    x[x==0] <- NA
  }
```
```r
### index of missings / non-missings
w <- is.na(x)
wN <- !is.na(x)
w2 <- apply(x, 1, function(x)
    length(which(is.na(x))))

if(method == "gmean"){
### mean imputation in Simplex:
geometricmean <- function(x) {
    if (any(na.omit(x == 0)))
        0
    else exp(mean(log(unclass(x)[is.finite(x) & x > 0])))
} 
gm <- apply(x, 2, function(x) {
    geometricmean(x[complete.cases(x)])
})
xmean <- x
for(i in 1:ncol(x)){
    xmean[,i, i] <- gm[i]
}
res <- list(xOrig=xcheck, xImp=xmean, criteria=0, iter=0, maxit=maxit, w=length(which(w)), wind=w)
} else if ( method=="meanClosed " ){
xmean <- x
impute <- function (x, what = c("median", "mean")) {
    what <- match.arg(what)
    if (what == "median") {
        retval <- apply(x, 2, function(z) {
            z[is.na(z)] <- median(z, na.rm = TRUE)
            z
        })
    } else if (what == "mean") {
        retval <- apply(x, 2, function(z) {
            z[is.na(z)] <- mean(z, na.rm = TRUE)
            z
        })
    } else {
        stop("what' invalid")
    }
    retval
} xmean <- impute(xmean)
res <- list(xOrig=xcheck, xImp=xmean, criteria=0, iter=0, maxit=maxit, w=length(which(w)), wind=w)
} else{
###sort the columns of the data according to the amount of missings in the variables
indM <- sort(apply(x,2,function(x) length(which(is.na(x)))),index.return=TRUE, decreasing=TRUE)$ix
cn <- colnames(x)

### first step - replace all NAs with values with 'nearest neighbour' algorithm
#if(init="NN"){
# library(templdistC)
# x <- templdistC(x)
#}
```
if(init=="KNN"){
  x <- impKNNa(x, k=k, metric="Aitchison", normknn=TRUE)$xImp
  #Aitchison
}
if(init=="KNNclosed"){
  x <- impKNNa(x, k=k, metric="Euclidean")$xImp
}
if(init=="roundedZero"){
  x[is.na(x)] <- 0.001
}

#x=acomp(x) #Aitchison compositions (for ilr)
#x2 <- acomp(xcheck) # with missings

##PCA algorithmus
it = 0
criteria <- 10000000
error <- rep(0, ncol(x))

# start the iteration
## require(StatDA)
## ternary(acomp(x))
# plot(ilr(x[w2==0,]), xlim=c(-5,5), ylim=c(-8,0.5))
# points(ilr(x[w2>0,]), col=gray(0.9), pch=3)
# gr <- seq(0.7,0.3, length.out=8)
while(it <= maxit & criteria >= eps){
  xold <- x
  it = it+1
  for(i in 1:ncol(x)){
    #change the first column with that one with the highest amount of NAs
    #in the step
    xNA = x[,indM[i]]
    x1 = x[,1]
    x[,1] = xNA
    x[,indM[i]] = x1
    if (closed == FALSE ) xilr = ilr(x) else xilr = x

    #apply the PCA algorithm -> ximp
    ind <- cbind(w[, indM[i]], rep(FALSE, dim(w)[1]))
    if(method == "classical" | method == "mcd" | method == "gridMAD"){
      xilr <- impPCA(xilr, indexMiss = ind, eps = 1,
                     indexObs=!ind, method = method)
    } #if( method == "em" ){ #
      s <- prelim.norm(as.matrix(xilr))
      thetahat <- em.norm(s, showitvs=FALSE)
      # xilr <- inp.norm(s, thetahat, as.matrix(xilr))
    #}
    # if( method == "lls" ){ #
    #  xilr <- suppressWarnings(llsImpute(xmiss, 3, verbose = FALSE)$completeObs)
    #}

    if(method == "ltsReg" | method == "lm"){
      #beta=ltsReg(xilr[,1]-xilr[,2],xilr)$coefficients
      xilr <- data.frame(xilr)
      c1 <- colnames(xilr)[1]
      colnames(xilr)[1] <- "V1"
      reg1 = get(method)(V1 ~ ., data=xilr)
      colnames(xilr)[1] <- c1
      #imp = cbind(rep(1, nrow(xilr)), xilr[,1])
      xilr[w[, indM[i]], 1] <- fitted(reg1)[w[, indM[i]]]
      xilr[w[, indM[i]], 1] <- fitted(reg1)[w[, indM[i]]]
    }
  }
}

© http://ameli.surveystatistics.net/ - 2011
if (method == "ltsReg2") {
  xilr <- data.frame(xilr)
c1 <- colnames(xilr)[1]
colnames(xilr)[1] <- "V1"
reg1 = ltsReg(V1 ~ ., data = xilr)
imp = as.matrix(cbind(rep(1, nrow(xilr)), xilr[,-1]))  ## %*% reg1$coef
colnames(xilr)[1] <- cl
  
### imp = cbind(rep(1, nrow(xilr)), xilr[, -1]) %*% reg1$coef
xilr[w[, indM[i]], 1] <- fitted(reg1)[w[, indM[i]]]
error[indM[i]] <- noise*sd(xilr[,1])#sqrt(mad(xilr[,1]))

### imp[w[, indM[i]]] + rnorm(length(imp[w[, indM[i]]]), 0, sd=0.5*sqrt(mad(xilr[,1])))
}
if (method == "roundedZero") {
  phi <- ilr(cbind(rep(1[indM[i]], nrow(x[])), x[-i, drop=FALSE]))[,1]
  xilr <- data.frame(xilr)
c1 <- colnames(xilr)[1]
colnames(xilr)[1] <- "V1"
reg1 = lm(V1 ~ ., data = xilr)
yhat2 <- predict(reg1, new.data = xilr[-i])
  
### colnames(xilr)[1] <- c1
#s <- sd(xilr[,1], na.rm=TRUE)
#ex <- phi - yhat/s
#yhat2 <- yhat - s*dnorm(ex)/pnorm(ex)
  
xilr[w[, indM[i]], 1] <- ifelse(yhat2[w[, indM[i]]] <= phi[w[, indM[i]]], phi[w[, indM[i]]], yhat2[w[, indM[i]]])
}
  
#if( method == "rf" ) {
#  xilr[w[, indM[i]], 1] <- NA
#  reg1 <- rfImpute(xilr[,i] ~ xilr[-i,], data=xilr)
#  xilr[w[, indM[i]], 1] <- reg1[w[, indM[i]]]
#}  

if( closed == FALSE )
  x=invilr(xilr) else x=xilr
  
# return the order of columns
xNA=x[,1]
x1=x[,indM[i]]
x[,1]=xNA
x[,indM[i]]=xNA


}

criteria <- sum((((xold - x)/x)^2, na.rm=TRUE)  ##sum(abs(as.matrix(xold) - as.
  matrix(x)), na.rm=TRUE)  ##DIRTY: (na.rm=TRUE)
# print(paste(method, ",", it, ",", "criteria="",round(criteria,3)))
if( closed == FALSE ) colnames(x) <- colnames(xcheck)
}

if( method == "ltsReg2") {  # finally, add an error for method ltsReg2
for( i in 1:ncol(x)){
  xNA=x[,indM[i]]
x1=x[,1]
x[,1]=xNA
x[,indM[i]]=x1
  
if( closed == FALSE ) xilr=ilr(x) else xilr=x
  
ind <- cbind(w[, indM[i]], rep(FALSE, dim(w)[1]))
xilr <- data.frame(xilr)
c1 <- colnames(xilr)[1]
colnames(xilr)[1] <- "V1"
reg1 = ltsReg(V1 ~ ., data=xilr)
imp = as.matrix(cbind(rep(1, nrow(xilr)), xilr[,-1]))  ## %*% reg1$coef
  
colnames(xilr)[1] <- c1
xilr[w[, indM[i]], 1] <- xilr[w[, indM[i]], 1] +

AMELI-WP4-D4.1
```r
rnorm(length(which(w[, indM[i]])), 0, sd=error[indM[i]])
xilr <- data.frame(xilr)
if( closed == FALSE ) x=invilr(xilr) else x=xilr
xNA=x[,1]
x1=x[,indM[ii]]
x[,1]=x1
x[,indM[ii]]=xNA
}
}
res <- list(xOrig=xcheck, xImp=x, criteria=criteria, iter=it,
maxit=maxit, w=length(which(w)), wind=w)
}
class(res) <- "imp"
invisible(res)
}

impPCA <- function(x, indexMiss, indexObs, method="classical", eps=0.5,
all.obs=FALSE, P=dim(x)[2], maxit=100)
### x ... Matrix or data.frame with missings
### method ... mcd, classical, gridMAD
### eps ... convergence criteria
### all.obs ... TRUE, if the whole observation should be replaced with the pca-estimate
###
### Author: Matthias Templ
### Licence: GPL 2.0

cm <- colMeans(x, na.rm=TRUE) ## fuer Ruecktransf.
csd <- sd(x, na.rm=TRUE) ## fuer Ruecktransf.
x <- apply(x, 2, function(x) (x - mean(x, na.rm=TRUE))/sd(x, na.rm=TRUE))

### PCA, Iteration:
d <- 1000000
it=0
while(d > eps & it <= maxit){
it=it+1
if( method == "mcd" ){ xMcd <- covMcd(x)
  p <- princomp(x, covmat=xMcd)
} else if( method == "classical" ){ p <- princomp(x) }

# if( method == "gridMAD" ){ p <- PCAgrid(x, method="mad", k=ncol(x)) }

xneu <- p$sco[,1:(P-1)] %*% t(p$load[,1:(P-1)]) # (p$load[,1:P]) # # rep(1,dim(x)[1])%*% t(xMcd$center) # p-dim???
d <- sum(abs(x[indexMiss] - xneu[indexMiss]))
if(all.obs == TRUE) x[indexMiss[,1],] <- xneu[indexMiss[,1],] else x[indexMiss] <- xneu

# print(d)
}

### Ruecktrans:
for( i in 1: dim(x)[2] ){
x[,i] <- (x[,i] * csd[i]) + cm[i]
}
return(x)

impKNNa <- function(x, method="knn", k=3,
metric="Aitchison", agg="median", primitive=FALSE, normknn=TRUE, das=FALSE, adj="median")
{
### Coda Version 2
### Nearest Neighbor imputation algorithm of Missing values
### in compositional data using robust methods
```
### Author: Matthias Templ
### Licence: GPL 2.0

```r
x <- as.matrix(x)
class(x) <- "matrix"
N <- dim(x)[1]
P <- dim(x)[2]
if(any(rowSums(is.na(x)) == P)) stop("One or more observations consist of only NA's")
xOrig <- x
xOrig[!is.na(x)] <- x
w <- !is.na(x)
w2 <- !is.na(x)
if(metric == "Euclidean" & primitive == FALSE)
  findknn <- function(x, i, j){
    ## find knn
    m1 <- which(!is.na(x[i,]))
    mi <- which(!is.na(x[,j, drop = FALSE]) & !is.na(rowSums(x[,m1, drop = FALSE])))
    d <- rowSums(t((t(x[mi, m1, drop = FALSE]) - x[i, m1])^2))
    names(d) <- mi
    wA <- which(d <= quantile(d, k / N))
    w <- as.numeric(names(wA))
    list(knn = w, m1 = m1)
  }
}
if(metric == "Euclidean" & primitive == TRUE)
  findknn <- function(x, i, j){
    ## find knn
    m1 <- which(!is.na(x[i,]))
    mi <- which(!is.na(x[, j, drop = FALSE]))
    d <- rowSums(t((t(x[mi, m1, drop = FALSE]) - x[i, m1])^2, na.rm = TRUE))
    names(d) <- mi
    wA <- which(d <= quantile(d, k / N))
    w <- as.numeric(names(wA))
    list(knn = w, m1 = m1)
  }
}
if(metric == "Aitchison" & primitive == FALSE)
  findknn <- function(x, i, j){
    m1 <- which(!is.na(x[i,]))
    mi <- which(c(!is.na(x[, j, drop = FALSE])) & c(!is.na(rowSums(x[, m1, drop = FALSE])))
    xclr <- clr(rbind(x[mi, m1, drop = FALSE], x[i, m1]))$x.clr
    d <- rowSums(abs(t(xclr[-nrow(xclr),]) - c(xclr[-nrow(xclr),])))
    names(d) <- mi
    wA <- which(d <= quantile(d, k / length(d)))
    w <- as.numeric(names(wA))
    list(knn = w, m1 = m1)
  }
}
if(metric == "Aitchison" & primitive == TRUE & das == TRUE)
  findknn <- function(x, i, j){
    m1 <- which(!is.na(x[i,]))
    mi <- which(c(!is.na(x[, j, drop = FALSE]))
    da <- function(x, y){
      d <- 0
      p <- length(x)
      for(i in 1:p-1){
        for(j in (i+1):p){
          d <- d + (log(x[i]/x[j]) - log(y[i]/y[j]))^2
        }
      }
      d = d / p
      d
    }
    ref <- x[i, m1]
    d <- apply(x[mi, m1, drop = FALSE], 1, function(x){da(x=x, y=ref)})
    names(d) <- mi
    wA <- which(d <= quantile(d, k / length(d)))
    w <- as.numeric(names(wA))
    list(knn = w, m1 = m1)
  }
}
```
if(metric=="Aitchison" & primitive==FALSE & das == TRUE){
findknn <- function(x, i, j){
  mi <- which(!is.na(x[i,]))
  mi <- which(c(!is.na(x[,j]), drop=FALSE)) & c(!is.na(rowSums(x[,mi, drop=FALSE])))
  da <- function(x,y){
    d <- 0
    p <- length(x)
    for(i in 1:(p-1)){
      for(j in (i+1):p){
        d <- d + (log(x[i]/x[j]) - log(y[i]/y[j]))^-2
      }
    }
    d=d/p
    d
  }
  ref <- x[i,m1]
  d <- apply(x[mi, m1, drop=FALSE], 1, function(z){da(x=z, y= ref )})
  names(d) <- mi
  wA <- which(d <= quantile(d, k/length(d)))
  w <- as.numeric(names(wA))
  list(knn =w, m1=m1)
}
if(metric=="Aitchison" & primitive==TRUE ){
findknn <- function(x, i, j){
  mi <- which(!is.na(x[i,]))
  a <- c(!is.na(x[,j, drop=FALSE]))
  b <- ifelse(rowSums(is.na(x,-j)) == 0, TRUE , FALSE )
  mi <- which(a & b)
  xclr <- clr(rbind(x[mi, m1, drop=FALSE], x[i, m1]))$x.clr
  d <- rowSums(t(abs(t(xclr[rowSums(xclr),])) - c(xclr[rowSums(xclr),]),)), na.rm=TRUE)
  #d[d == 0] <- NA ## dirty
  names(d) <- mi
  wA <- which(d <= quantile(d, k/length(d), na.rm=TRUE))
  w <- as.numeric(names(wA))
  list(knn =w, m1=m1)
}
}
if(metric=="Euclidean"){
imp <- function(x, i, j){
  ## workhorse: do the imputation
  knn <- findknn(x, i, j)
  get(agg)(x[knn,j], na.rm=TRUE) #median
}
}
if(metric=="Aitchison" & normknn == FALSE){
imp <- function(x, i, j){
  ## workhorse: do the imputation
  knn <- findknn(x, i, j)
  ## median, normalization:
  #fac <- knn$xclriSum/knn$medKnnSum
  ##median(x[knn$knn, j])*fac # fac anders berechnen
  medSum <- get(adj)(apply(x[knn$knn, knn$m1, drop=FALSE], 2, function(x,...)(get(agg)(x, na.rm=TRUE)) ), na.rm=TRUE)
  xSum <- get(adj)(x[i, knn$m1, drop=FALSE], na.rm=TRUE)
  fac <- xSum/medSum
  get(agg)(x[knn$knn, knn$m1], na.rm=TRUE)*fac #median
}
}
if(metric=="Aitchison" & normknn == TRUE){
imp <- function(x, i, j){
  ## workhorse: do the imputation
  knn <- findknn(x, i, j)
  ## normalization:
  #xSum <- get(adj)(x[i, knn$m1, drop=FALSE], na.rm=TRUE)
  xSum <- get(adj)(x[i, knn$m1, drop=FALSE], na.rm=TRUE)
  if( length(knn$knn) == 1){
    #knnSum <- sum(x[knn$kmn, knn$m1], na.rm=TRUE)
    knnSum <- get(adj)(x[knn$kmn, knn$m1], na.rm=TRUE)
  }
}
}
© http://ameli.surveystatistics.net/ - 2011
```r
} else{
  # knnSum <- rowSums(x[knn$knn, knn$m1, drop=FALSE], na.rm=TRUE)
  knnSum <- apply(x[knn$knn, knn$m1, drop=FALSE], 1, function(x,...){get(agg)(x, na.rm=TRUE)})
}

fac <- xSum/knnSum
get(agg)(x[knn$knn, j] * fac, na.rm=TRUE)
## statt Mean das geom. Mittel?
}

# if(metric == "Aitchison" & ratios == TRUE){
#  knn <- findknn(x, i, j)
#  p <- ncol(x)
#  y <- apply(apply(x[knn$knn, , drop = FALSE], 1, function(x){x[2:p]/x[1]}),1,median)
#  x[i,j] <- 1
#  x[i,-j] <- y
#
#} # if(metric="Aitchison" & unweighted == TRUE){
#  imp <- function(x, i, j){
#    knn <- findknn(x, i, j)
#    x[i,j] <- get(agg)(x[knn$knn, j])
#  }
#} #}

for(i in 1:N){
  for(j in 1:P){
    if(is.na(x[i,j])){
      x[i,j] <- imp(xmiss, i, j)
    }
  }
}

w <- is.na(xOrig)
colnames(x) <- colnames(xOrig)
res <- list(xOrig = xOrig, xImp=x, criteria=NULL, iter=NULL, w=length(which(w)), wind=w, metric=metric)
class(res) <- "imp"
res
}

'invilr' <- function(x.ilr)
{
### inverse isometric log-ratio transformation
### Author: Peter Filzmoser, Karel Hron, Matthias Templ
### Licence: GPL 2.0
y=matrix(0,nrow=nrow(x.ilr),ncol=ncol(x.ilr)+1)
D=ncol(x.ilr)+1
y[,1]=sqrt((D-1)/D)*x.ilr[,1]
for (i in 2:ncol(y)){
  for (j in 1:(i-1)){
    y[,i]=y[,i]+x.ilr[,j]/sqrt((D-j+1)*(D-j))
  }
  for (i in 2:(ncol(y)-1)){
    y[,i]=y[,i]-sqrt((D-i)/(D-i+1))*x.ilr[,i]
  }
}
yexp=exp(y)
x.back=yexp/apply(yexp,1,sum) # * rowSums(derOriginaldaten)
invisible(x.back)
#return(yexp)
}

'ilr' <- function(x)
{
### isometric log-ratio transformation
### Author: Matthias Templ, Peter Filzmoser

AMELI-WP4-D4.1
```
### Licence: GPL 2.0

```r
x.ilr = matrix(NA, nrow = nrow(x), ncol = ncol(x) - 1)
D = ncol(x)
for (i in 1:ncol(x.ilr)) {
  x.ilr[, i] = sqrt((D - i) / (D - i + 1)) * log(((apply(as.matrix(x[, (i + 1):D], drop = FALSE)), 1, prod) - (1 / (D - i))) / (x[, i]))
}
invisible(x.ilr)
}
```

```c
#include <R.h>
# include <math.h>
/*
 * Author: Matthias Templ */
/* Licence: GPL 2.0 */

void da(double *matOrig, double *matImp, int *dims, double *rowDists, double *distance) {
  int rows = dims[0];
  int cols = dims[1];
  /* double result; */
  /* int N = rows * cols; */
  int i, j, k;
  /* float erg; */

  /* Fuer alle Zeilen */
  for (i = 0; i < rows; i++) {
    rowDists[i] = 0.0;
    /* ueber die entsprechenden Spalten */
    for (j = (i * cols); j < (i * cols) + cols - 1; j++) {
      for (k = j + 1; k < (i * cols) + cols; k++) {
        rowDists[i] = rowDists[i] + pow((log(matOrig[j] / matOrig[k]) - log(matImp[j] / matImp[k])), 2);
      }
    }
    /* Gesamtdistanz aufsummieren */
    distance[0] = distance[0] + sqrt(rowDists[i] / cols);
  }
  /* result = distance[rows];
  return result; */
  /* return distance[rows]; das funkt leider nicht, brauche nur einen Wert zurueckgeben! */
}
```

© http://ameli.surveystatistics.net/ - 2011
Chapter 11
Sequential Robust Imputation

`prop5b` <- function(x, eps=0.01, maxit=100,mixed=NULL, step=FALSE, robust=FALSE, takeAll=TRUE,
noise=TRUE, noise.factor=1, force=FALSE,
robMethod="lmrob", force.mixed=TRUE, mi=1, addMixedFactors=FALSE)
{
  #### robust sequential model-based imputation
  ## Programme by Matthias Templ and Alexander Kowarik
  ## Licence: GPL 2.0
  ## this function is included in R-package VIM,
  ## available on CRAN
  `initialise` <-
  function(x){
    if(class(x) == "numeric") {x <- as.vector(impute(as.matrix(x), "mean"))}
    if(class(x) == "integer") {x <- as.vector(impute(as.matrix(x), "mean"))}
    if(class(x) == "factor") {x <- as.character(x)
      x[which(is.na(x))] <- names(which.max(table(x)))
      x <- as.factor(x)}
    return(x)
  }

  require(nnet)
  require(e1071)
  require(robustbase)
  require(MASS)
  n <- nrow(x)
  anyNA <- function(X) any(is.na(X))
  Unit <- function(A) UseMethod("Unit")
  Unit.list <- function(A){ # Units a list of vectors into one vector
    a <- vector()
    for(i in 1:length(A)){
      a <- c(a,A[[i]])
    }
    levels(as.factor(a))
  }
  Inter <- function(A) UseMethod("Inter")
  Inter.list <- function(A){ # common entries from a list of vectors
    a <- Unit(A)
    TF <- rep(TRUE, length(a))
    for(i in 1:length(a)){
      for(j in 1:length(A)){
        TF[i] <- TF[i] && a[i] %in% A[[j]]
      }
    }
    levels(as.factor(a[TF]))
  }

  ### Matthias Templ, last modification 25. Nov. 2008
  ### x ... matrix or data.frame with missings
if(step&&robust)
  stop("robust stepwise not yet implemented")
factors <- vector()
for(i in 1:ncol(x)){
  factors <- c(factors,is.factor(x[,i]))
}
factors <- colnames(x)[factors]
N <- dim(x)[1]
P <- dim(x)[2]
VarswithNA <- vector()
### index fuer missing/non-missing
w2 = is.na(x)
for(i in seq(P)){
  if(anyNA(x[,i]))
    VarswithNA <- c(VarswithNA,i)
}
getM <- function(xReg, ndata, type, index,mixedTF,factors,step,robust,noise,noise.factor=1) {
  switch(type,
    numeric = useLM(xReg, ndata, index,mixedTF,factors,step,robust,noise,noise.factor ),
    factor = useMN(xReg, ndata, index,factors,step,robust),
    bin = useB(xReg, ndata, index,factors,step,robust)
  )
}
useLM <- function(xReg, ndata, wy,mixedTF,factors,step,robust,noise,noise.factor)
  factors <- Inter(list(colnames(xReg),factors))
  if(mixedTF){
    delFactors <- vector()
    if(length(factors)>0){
      for(f in 1:length(factors)){
        if(any(summary(xReg[,f])==0)){
          xReg <- xReg[-which(colnames(xReg)==f)]
          ndata <- ndata[-which(colnames(ndata)==f)]
          delFactors <- c(delFactors,f)
        }
      }
    }
    xReg1 <- xReg
    xReg1$y[xReg1$y!=0] <- 1
    if(!robust)
      glm.bin <- glm(y ~ . , data=xReg1, family="binomial")
    else{
      glm.bin <- glm(y ~ . , data=xReg1, family="binomial")
      if(force.mixed){
        if(exists("glm.bin"))
          rm(glm.bin)
        try(glm.bin <- glmrob(y ~ . , data=xReg1, family="binomial"),silent=TRUE)
        if(exists("glm.bin"))
          glm.bin$rank <- ncol(xReg)
        else
          glm.bin <- glm(y ~ . , data=xReg1, family="binomial")
        }
      else{
        glm.bin <- glmrob(y ~ . , data=xReg1, family="binomial")
        glm.bin$rank <- ncol(xReg)
      }
    }
  }
  # op <- options() #Alles auskommentiert, weil VGAM draussen!
  # options(show.error.messages=FALSE)
  # try(detach(package:VGAM))
  # options(op)
  if(step)
    glm.bin <- stepAIC(glm.bin,trace=-1)
  imp <- predict(glm.bin, newdata=ndata, type="response")
  imp[imp < 0.5] <- 0
  imp[imp >= 0.5] <- 1
  xReg <- xReg[xReg$y > 0.]
  factors2 <- factors[factors%in%delFactors]
  if(length(factors2)>0){
  }
Chapter 11. Sequential Robust Imputation

```r
for(f in 1:length(factors2)){
  if(any(summary(xReg[,factors2[f]])==0)){
    xReg <- xReg[,!which(colnames(xReg)==factors2[f])]
    ndata <- ndata[,!which(colnames(ndata)==factors2[f])]
  }
}
}
else{
  if(length(factors)>0){
    delFactors <- vector()
    for(f in 1:length(factors)){
      if(any(summary(xReg[,factors[f]])==0)){
        xReg <- xReg[,!which(colnames(xReg)==factors[f])]
        ndata <- ndata[,!which(colnames(ndata)==factors[f])]
        delFactors <- c(delFactors,factors[f])
      }
    }
    imp <- rep(1,nrow(ndata))
  }
}
## Two-Step
if(!robust){
  glm.num <- glm(y~.,data=xReg,family="gaussian")
  # cat(" not ROBUST!!!!!!!\n")
} else{
  if(exists("glm.num")){
    rm(glm.num)
    if(force){
      try(glm.num <- lmrob(y~.,data=xReg),silent=TRUE)
      if(!exists("glm.num")){
        try(glm.num <- rlm(y~.,data=xReg,method="MM"),silent=TRUE)
        if(!exists("glm.num")){
          glm.num <- glm(y~.,data=xReg,method="gaussian")
        }
      }
    }
  } else{
    if(robMethod=="lmrob"){
      # cat(" ROBUST!!!!!!!\n")
      glm.num <- lmrob(y~.,data=xReg)
    } else if(robMethod=="lqs"){
      glm.num <- lqs(y~.,data=xReg)
    } else{
      glm.num <- rlm(y~.,data=xReg,method=robMethod)
    }
  }
  # op <- options()# Alles auskommentiert, weil VGAM draussen
  # options(show.error.messages=FALSE)
  # try(detach(package:VGAM))
  # options(op)
  if(step){
    glm.num <- stepAIC(glm.num,trace=1)
  }
  if(noise){
    if(!robust){
      consistencyFactor <- sqrt((nrow(ndata[,imp==1,drop=FALSE])/n + 1)*((n-1)/n))
      p glm.num <- predict(glm.num,newdata=ndata[,imp==1,drop=FALSE],se.fit=TRUE)
      imp2 <- p glm.num$fit*noise.factor*rnorm(length(p glm.num$fit),0,p glm.num$scale*residual.scale*consistencyFactor)
    } else{
      consistencyFactor <- sqrt((nrow(ndata[,imp==1,drop=FALSE])/n + 1)*((n-1)/n))
      glm.num <- glm.num
      p glm.num <- predict(glm.num,newdata=ndata[,imp==1,drop=FALSE])
    # print(glm.num$se)
    # print((consistencyFactor*glm.num$scale*sum(glm.num$weights)/n))
    imp2 <- p glm.num + noise.factor*rnorm(length(p glm.num),0,glm.num$se*consistencyFactor)
  }
```

AMELI-WP4-D4.1
} otherwise
imp2 <- predict(glm.num, newdata=ndata[imp==1,drop=FALSE])
imp[imp==1] <- imp2
return(imp)

# library(VGAM, warn.conflicts = FALSE, verbose=FALSE)

useMN <- function(xReg, ndata, wy, factors, step, robust){
factors <- Inter(list(colnames(xReg),factors))
if(length(factors)>0){
  for(f in 1:length(factors)){
    if(any(summary(xReg[,factors[f]])==0)){
      xReg <- xReg[,which(colnames(xReg)==factors[f])]
      ndata <- ndata[,which(colnames(ndata)==factors[f])]
    }
  }
}
# multinom statt VGAM, wenn wieder zurueck auf VGAM, muessen alle
# library(VGAM, warn.conflicts = FALSE, verbose=FALSE)

# TODO: Faktoren mit 2 Levels und nicht Levels 0 1, funktionieren NICHT!!!!

if(length(factors)>0){
  for(f in 1:length(factors)){
    if(any(summary(xReg[,factors[f]])==0)){
      xReg <- xReg[,which(colnames(xReg)==factors[f])]
      ndata <- ndata[,which(colnames(ndata)==factors[f])]
    }
  }
}
if(!robust)
  glm.bin <- glm(y ~ ., data=xReg, family="binomial")
else{
  glm.bin <- glm(y ~ ., data=xReg, family="binomial")
  if(!exists("glm.bin"))
    rm(glm.bin)
  try(glm.bin <- glmrob(y ~ ., data=xReg, family="binomial"),silent=TRUE)
  if(!exists("glm.bin"))
    glm.bin$rank <- ncol(xReg)
  else
    glm.bin <- glm(y ~ ., data=xReg, family="binomial")
}
op <- options() # Alles auskommentiert, weil VGAM draussen!!!!
options(show.error.messages=FALSE)
try(detach(package:VGAM))
options(op)
if(step)
  glm.bin <- stepAIC(glm.bin, trace=-1)
imp <- predict(glm.bin, newdata=ndata, type="response")
imp[imp < 0.5] <- 0
imp[imp >= 0.5] <- 1
# library(VGAM, warn.conflicts = FALSE, verbose=FALSE)
return(imp)
}

***************
### initialisiere
for( j in 1:ncol(x) ) {
 x[,j] <- initialise(x[,j])
}

# print(x)
mixedTF <- FALSE

### außere Schleife:
d <- 99999
it <- 0
while(d > eps && it < maxit){
  it = it + 1
  xSave <- x
  ### innere Schleife:
  for(i in VarswithNA){
    flush.console()
    yPart <- x[, i, drop=FALSE]
    wy <- which(w2[,i])
    # Den ganzen Datensatz
    # print (wy)
    xPart <- x[, -i, drop=FALSE]
    ## Start Additional xvars for mixed vars
    if(! is.null(mixed) && addMixedFactors){
      mixedIndex <- which(names(xPart) %in% mixed)
      for(i in 1:length(mixedIndex)){
        namenew <- paste(names(xPart)[mixedIndex[i]], "ADDMIXED", sep="")
        xPart[,namenew] <- as.numeric(xPart[,mixedIndex[i]]==0)
        #cat(namenew)
      }
    }
    # End additional xvars for mixed vars
    if(! takeAll){
      dataForReg <- data.frame(cbind(yPart[-wy ,], xPart[-wy ,])) ## part, wo in y keine
      missings
    } else{
      dataForReg <- data.frame(cbind(yPart, xPart))
    }
    if(! is.null(mixed)){
      if(names(x)[i] %in% mixed){
        mixedTF <- TRUE
      } else{
        mixedTF <- FALSE
      }
    }
    colnames(dataForReg)[1] <- "y"
    new.dat <- data.frame(cbind(rep(1, length(wy)), xPart[wy ,, drop=FALSE]))
    #i <- i
    #colnames(new.dat)[1] <- "y"
    if( class(dataForReg$y) == "numeric" ) meth = "numeric" else if( class(dataForReg$y) == "factor" 
    & length(levels(dataForReg$y))==2) meth = "bin" else meth = "factor"
    ## replace initialised missings:
    if(length(wy) > 0) x[wy,i] <- getM(xReg=dataForReg, ndata=new.dat[-,1,drop=FALSE],
    type=meth, index=wy, mixedTF=mixedTF, factors=factors, step=step, robust=robust,
    noise=FALSE)
  }
  d = sum((xSave - x)^2 , na.rm=TRUE) #to do: Faktoren anders behandeln.
  flush.console()
  #print(paste("it ",it," , Wert ",d))
  #print(paste("eps", eps))
  #print(paste("test":",d > eps))
  }
if( it > 1 ){  
  d <- sum((xSave - x)^2, na.rm=TRUE)
  if( it < maxit ) {print(paste(d, "," , eps," = eps"); print(paste(" finished after", it, "iterations"))}
  if( it == maxit ) {print("not converged...")}
}
# A last run with building the model and adding noise...
if(noise&&mi==1){
  for(i in seq(P)){
    flush.console()
    yPart <- x[, i, drop=FALSE]
    wy <- which(w2[,i])

    # Den ganzen Datensatz
    #print(wy)
    xPart <- x[, -i, drop=FALSE]
    if(!takeAll){
      dataForReg <- data.frame(cbind(yPart[-wy,], xPart[-wy,])) ## part, wo in y keine missings
    }else{
      dataForReg <- data.frame(cbind(yPart, xPart))
    }
    if(!is.null(mixed)){
      if(names(x)[i] %in% mixed){
        mixedTF <- TRUE
      }else{
        mixedTF <- FALSE
      }
    }
    colnames(dataForReg)[1] <- "y"
  }
}
new.dat <- data.frame(cbind(rep(1,length(wy)), xPart[wy,drop=FALSE]))
#i <-<  i
#colnames(new.dat)[1] <- "y"

if(class(dataForReg$y) == "numeric") meth = "numeric" else if(class(dataForReg$y) == "factor" & length(levels(dataForReg$y)) == 2) meth = "bin" else meth = "factor"
# replace initialised missings:
if(length(wy) > 0) x[wy,i] <- getM(xReg=dataForReg, ndata=new.dat[-1,drop=FALSE], type=meth, index=wy,mixedTF=mixedTF,factors=factors,step=step,robust=robust, noise=TRUE,noise.factor=noise.factor)
}
## End NOISE
## Multiple Imputation
if(mi>1&&!noise){
  cat("Noise option is set automatically to TRUE")
  noise <- TRUE
}
if(mi>1){
  xSave1 <- x
  for(m in 1:mi){
    for(i in seq(P)){
      flush.console()
      yPart <- x[, i, drop=FALSE]
    }
  }
}
# Den ganzen Datensatz
#print(wy)
}
if(class(dataForReg$y) == "numeric") meth = "numeric" else if(class(dataForReg$y) == "factor" & length(levels(dataForReg$y)) == 2) meth = "bin" else meth = "factor"

## replace initialised missings:
if(length(wy) > 0) x[wy,i] <- getM(xReg=dataForReg, ndata=new.dat[,-1,drop=FALSE]
, type=meth, index=wy, mixedTF=mixedTF, factors=factors, step=step, robust=robust, noise=TRUE, noise.factor=noise.factor)

mimp[[m]] <- x
x <- xSave1
} else
x <- mimp
}

## End Multiple Imputation
invisible(x)