EUROPEAN COMMISSION
Community Research

SEVENTH FRAMEWORK PROGRAMME

AMELI
Advanced Methodology for European Laeken Indicators

Deliverables 10.3

R packages plus manual

Version: 2011

Matthias Templ, Andreas Alfons, Peter Filzmoser, Monique Graf, Beat Hulliger, Jan-Philipp Kolb, Risto Lehtonen, Ralf Münnich, Desislava Nedyalkova, Tobias Schoch, Ari Veijanen, Stefan Zins

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 to Deliverable 10.3

Chapter 1: Matthias Templ, Andreas Alfons, Monique Graf, Beat Hulliger, Risto Lehtonen, Tobias Schoch, Ari Veijanen, Stefan Zins.

Chapter 2: Andreas Alfons, Matthias Templ, Peter Filzmoser.

Chapter 3: Andreas Alfons, Matthias Templ, Peter Filzmoser.

Chapter 4: Matthias Templ, Andreas Alfons.

Chapter 5: Matthias Templ, Andreas Alfons.

Chapter 6: Andreas Alfons, Matthias Templ, Peter Filzmoser, Josef Holzer.

Chapter 7: Matthias Templ, Andreas Alfons.

Appendix A1: Andreas Alfons.

Appendix A2: Andreas Alfons, Stefan Kraft.

Appendix A3: Matthias Templ, Andreas Alfons, Alexander Kowarik.

Appendix A4: Andreas Alfons, Josef Holzer, Matthias Templ.

Appendix A5: Monique Graf, Desislava Nedyalkova.

Appendix A6: Beat Hulliger, Tobias Schoch.

Appendix A7: Beat Hulliger, Tobias Schoch.

Appendix A8: Beat Hulliger, Tobias Schoch.

Appendix A9: Tobias Schoch

Main responsibility

The AMELI Team

Evaluators

Internal expert: General Assembly

Aim and objectives of deliverable 10.3

This final report will comprise the computer codes. The codes are splited into several R-packages. In addition specific R code published in the annex of Deliverable 7.1. (Hulliger et al., 2011a) and other deliverables such as Deliverable 2.2 (Lehtonen et al., 2011).

This deliverable is designed to be an enhancement to the usual R documentation manuals that are included in the appendix additionally. Such R documentation manuals are the proposed way for documenting R packages by using well-structured help files written in the R documentation (Rd) format. These R-help files are structured into sections explaining the paramters of the function their usage and includes some details on the implementation of the corresponding functions. Typically, the help files contain code in two sections: usage and examples.

However, the Rd file format was designed for documentation of single R objects (such as functions, data sets, methods, classes, ...). It is not intended for demonstrating the interaction of multiple functions in a package (Leisch, 2003). For this task the concept of package vignettes is designed. A package vignette is a document explaining parts or all of the functionality of a package in a more informal way than the strict format of reference help pages. We use the concept of package vignettes to show the application of the developed R code to the EU-SILC data. A package vignette consists of (and is generated from) R-code and LaTeX source code. This code is open-source and it is available in the `inst/doc` directory of the corresponding package.

The package vignettes in this deliverable concentrate to show applications of the packages to EU-SILC data. Potential users of the packages should easily see how the functions are applied to complex estimation purposes and exploring and visualization of the results.

The deliverable is stuctured as follows. A brief description of all packages is provided in Chapter 1. Package vignettes are available for the `simFrame`, the `simPopulation`, the `VIM` and the `laeken` package and included as chapters 2–7 of this deliverable. Further details about the functions applied in the vignettes are provided in the appendix where all function arguments as well as code for further examples are listed in form of traditional R packages manuals.

Contents

1 References and description of R packages **3**
- 1.1 SimFrame . . . 3
- 1.2 SimPopulation . . . 3
- 1.3 VIM . . . 4
- 1.4 laeken . . . 4
- 1.5 GB2 . . . 5
- 1.6 Robust non-parametric QSR estimation . . . 5
- 1.7 MODI . . . 5
- 1.8 rsae: Robust Small Area Estimation . . . 5
- 1.9 Specific R-Code . . . 6
  - 1.9.1 Work Package 2 . . . 6
  - 1.9.2 Work Package 3 . . . 6
  - 1.9.3 Work Package 4 . . . 6
  - 1.9.4 Work Package 8 . . . 7

Bibliography **12**

2 Applications of Statistical Simulation Using `simFrame` **17**
- 2.1 Introduction . . . 17
- 2.2 Application of different simulation designs to EU-SILC . . . 18
  - 2.2.1 Basic simulation design . . . 19
  - 2.2.2 Using stratified sampling . . . 19
  - 2.2.3 Adding contamination . . . 21
  - 2.2.4 Performing simulations separately on different domains . . . 22
  - 2.2.5 Using multiple contamination levels . . . 24
  - 2.2.6 Inserting missing values . . . 26
  - 2.2.7 Parallel computing . . . 28
- 2.3 Conclusions . . . 30

3 Simulation of EU-SILC Population Data Using `simPopulation` **33**
- 3.1 Introduction . . . 33
- 3.2 Wrapper function for EU-SILC . . . 34
- 3.3 Step by step instructions and diagnostics . . . 35
- 3.4 Conclusions . . . 42

4 An application of `VIM` to EU-SILC data **43**
- 4.1 The graphical user interface of `VIM` . . . 43
  - 4.1.1 Handling data . . . 43
  - 4.1.2 Selecting variables . . . 45
  - 4.1.3 Selecting plots . . . 45

4.2 An application to EU-SILC data . . . 45
4.2.1 Univariate plots . . . 47
4.2.2 Bivariate plots . . . 48
4.2.3 Multivariate plots . . . 48
4.2.4 Other plots . . . 50
4.3 Fine tuning . . . 51
4.4 Interactive features . . . 51
4.5 Summary . . . 52

5 Standard Methods for Social Exclusion Indicators in package `laeken` 55
5.1 Introduction . . . 55
5.2 Basic design of the package . . . 57
5.2.1 Class structure . . . 58
5.3 Calculation of the equivalized disposable income . . . 59
5.4 Weighted median and quantile estimation . . . 61
5.5 Indicators on social exclusion and poverty . . . 62
5.5.1 At-risk-at-poverty rate . . . 63
5.5.2 Quintile share ratio . . . 65
5.5.3 Relative median at-risk-of-poverty gap (by age and gender) . . . 66
5.5.4 Gini coefficient . . . 68
5.6 Extracting information using the `subset()` method . . . 69
5.7 Conclusions . . . 70

6 Robust Pareto Tail Modeling with package `laeken`. 73
6.1 Introduction . . . 73
6.2 Social exclusion indicators . . . 74
6.2.1 Quintile share ratio (QSR) . . . 75
6.2.2 Gini coefficient . . . 75
6.3 The Pareto distribution . . . 76
6.4 Finding the threshold . . . 77
6.4.1 Van Kerm's rule of thumb . . . 78
6.4.2 Pareto quantile plot . . . 78
6.4.3 Mean excess plot . . . 79
6.5 Estimation of the shape parameter . . . 81
6.5.1 Hill estimator . . . 82
6.5.2 Weighted maximum likelihood estimator . . . 82
6.5.3 Integrated squared error estimator . . . 84
6.5.4 Partial density component estimator . . . 85
6.6 Estimation of the indicators using Pareto tail modeling . . . 85
6.7 Conclusions . . . 87

7 Variance Estimation of Indicators using package `laeken` 91
7.1 Introduction . . . 91
7.2 General wrapper function for variance estimation . . . 92
7.3 Naive bootstrap . . . 93
7.4 Calibrated bootstrap . . . 95
7.5 Conclusions . . . 97

A1. simFrame Manual 100

A2. simPopulation Manual 216

A3. VIM Manual 250

A4. laeken Manual 309

A5. GB2 Manual 372

A6. Robust Horvitz-Thompson Estimation (RHT) 400

A7. BQSR, TQSR and MQSR 415

A8. MODI: Mulivariate Outlier Detection and Imputation 420

A9. rsae: Robust Small Area Estimation 435

Chapter 1

References and description of R packages

1.1 SimFrame

In order to simplify using common guidelines in simulation experiments, a software framework has been developed in the R package `simFrame` (Alfons et al., 2010b). It allows the use a wide range of simulation designs with a minimal effort of programming. In addition, the object-oriented implementation provides clear interfaces for further extensions.

Simulation studies in research projects such as AMELI require a precise outline. If different partners use, e.g., different contamination or missing data models, the results may be incomparable. A software framework for statistical simulation may thus contribute its share to avoid such problems. For this purpose, the R package `simFrame` (Alfons et al., 2010b) has been developed. The object-oriented implementation with S4 classes and methods (Chambers, 1998, 2008) gives maximum control over input and output and provides clear interfaces for user-defined extensions. Moreover, the framework allows a wide range of simulation designs to be used with only a little programming.

One of the main goals of the AMELI project is to improve the methodology for the indicators on social exclusion and poverty under typical data problems such as outliers and missing data. The package `simFrame` therefore allows to add certain proportions of outliers or non-response. In addition, depending on the structure of the simulation results, an appropriate plot method is selected automatically.

While the `simFrame` paper is published in Alfons et al. (2010b) and the basic structure is also outlined in Alfons et al. (2011a), a vignette shows the application of the package for EU-SILC. This package vignette is included in the Appendix (see also Alfons et al., 2010a).

1.2 SimPopulation

One aim of the AMELI project was to investigate robust estimation of the Laeken Indicators. For this purpose, Alfons et al. (2011d) developed a data generation framework, which is implemented in the R package `simPopulation` (Alfons and Kraft, 2010; Alfons et al., 2011d). Based on Austrian EU-SILC sample data, the synthetic population

AAT-SILC was generated with this framework (see Alfons et al., 2011b). AAT-SILC was designed to resemble a representative country. A further objective was that the population data should not contain any large outliers, as these are included in the samples during the simulations for full control over the amount of outliers (see Alfons et al., 2011a).

While the `simPopulation` is published in Alfons et al. (2011d), a package vignette in the appendix shows the application of the package.

1.3 VIM

Imputation of item non-responses in complex surveys has an effect on the final estimates of the indicators. One aim of the AMELI project was to develop robust methods for estimation (see Hulliger et al., 2011b) and to visualize the structure of microdata (see Templ et al., 2011a). Package `VIM` (Templ et al., 2011b) allows to explore the data with missing values and learn about the structure of the missing values. Visualisation methods such as modified parallel coordinate plots, mosaic plots, scatterplot matrices, etc., have to be modified to deal with missing values and to show their structure.

EM-based regression imputation algorithms are mainly used to impute missing values automatically, i.e. such methods are very helpful in hand of subject matter specialists who are not statistical experts. Since data virtually always comes with outlying observations, robust methods for statistical estimation of missing values should be used here. The implemented algorithm (see Templ et al., 2011c) again is able to deal with all data challenges like representative and non-representative outliers and a mixture of different distributed variables, for example.

The free and open-source R package `VIM` provides a graphical user interface for users having no experience with R.

More about `VIM` is shown in Templ et al. (2011a) and in the package vignette in the appendix.

1.4 laeken

One aim of the AMELI project was on estimation of social inclusion indicators. The methodology of estimating these indicators is implemented in package `laeken` (Alfons et al., 2011c). The package contains a subset of synthetically generated data for the European Union Statistics on Income and Living Conditions (EU-SILC), which is used in the code examples throughout the package. The package has an object-oriented design and different classes and subclasses are introduced.

In addition, robust semiparametric estimation (see Hulliger et al., 2011b) of social exclusion indicators is available. Special emphasis is thereby given to income inequality indicators, as the standard estimates for these indicators are highly influenced by outliers in the upper tail of the income distribution. This influence can be reduced by modeling the upper tail with a Pareto distribution in a robust manner.

Moreover, variance estimation methods are implemented in the package. To be more precise, it describes a general framework for estimating variance and confidence intervals of indicators under complex sampling designs. Currently, the package is focused on bootstrap approaches. While the naive bootstrap does not modify the weights of the bootstrap samples, a calibrated version allows to calibrate each bootstrap sample on auxiliary information before deriving the bootstrap replicate estimate.

The package vignettes are included in the appendix.

1.5 GB2

Package `GB2` (Graf and Nedyalkova, 2010) implements the methods described in Graf et al. (2011, Chapters 1-4). For the Generalized Beta distribution of the second kind (GB2) - density, distribution function, quantiles, moments are provided. Functions for the full log-likelihood, the profile log-likelihood and the scores are given. Formulae for various Laeken indicators under the GB2 are implemented. Package GB2 performs pseudo maximum likelihood estimation and non-linear least squares estimation of the model parameters and computes the design based variance of the parameters and the indicators by linearization. It provides various plots for the visualization and analysis of the results.

1.6 Robust non-parametric QSR estimation

The `rqsr` package is an implementation of the robust, non-parametric QSR estimators. Namely, it enables the user to compute `TQSR`, `SQSR`, `BQSR`, and `MQSR`. In addition, it includes a device to compute variance estimates for all variants of the robustified QSR. See Appendix for more details.

1.7 MODI

The `modi` package contains functions for robust multivariate outlier detection and imputation. The following detection methods are implemented: `BACON-EEM`, `TRC`, `GIMCD`, and `Epidemic Algorithm`. All algorithms can cope with both missing values and complex survey samples. Once the data have been processed by outlier-detection methods, one considers (robustly) imputing for the missing values and the declared outliers. The implemented imputation methods are `Gaussian imputation` (based on robustly estimated location and scatter), `Nearest Neighbor Imputation`, and `Reverse Epidemic Algorithm`. See Appendix for more details.

1.8 rsae: Robust Small Area Estimation

The `rsae` package offers a general framework to robustly estimate area-level- and unit-level small area estimation (SAE) models. Once a particular model has been set up, it can be

fitted by various robust methods (and also maximum likelihood). The `rsae` consists of two fitting modes: "default mode" and "safe mode". The latter involves a high-breakdown-point regression estimator initialization and uses several numerical checks whether the iteration-specific estimates behave well. Currently, only Huber-type M-estimation is implemented. This method has assured super-linear convergence (given that (1) the amount of contamination is strictly below the breakdown point and (2) the model is properly specified). The high-breakdown-point S-estimator for mixed-level models will be included in the next release. Once the parameters of the Gaussian core model have been robustly estimated, we consider robustly predicting the random effects and the small-area means. Further, the package is shipped with several useful utility functions. See Appendix for more details.

1.9 Specific R-Code

1.9.1 Work Package 2

R functions have been programmed for small area estimation (SAE) of indicators on poverty and social exlusion. The indicators include at-risk-of poverty rate, the Gini coefficient, relative median at-risk-of poverty gap and quintile share ratio (S20/S80 ratio). Design-based estimators include direct estimators that do not use auxiliary data. The more advanced indirect model-assisted, model-based and composite estimators use auxiliary data at unit level or at aggregated level and generalized linear mixed models. We have fitted most of the mixed models with R functions `nlme` and `glmer` (package `lme4`). In addition, R function multinom of package `nnet` has been used. Technical description of SAE methodology is in Lehtonen et al. (2011). Annex 1 (Manual of R codes) of Lehtonen et al. (2011) includes a more detailed description of R codes. The R program codes can be found in separate AMELI deliverable files.

1.9.2 Work Package 3

Variance estimation with the linearized variance estimators, was done with the help of the `survey` package (cf. Tille and Matei, 2011), the definition of the necessary `survey.design` objects can be found in Hulliger et al. (2011a, section 9). The functions used to compute both point and variance estimates for the linearized estimators are given in the appendix of Hulliger et al. (2011a) (see *R Functions for Computing Point and Variance Estimates*).

1.9.3 Work Package 4

Specific code for outlier detection of semi-continuous variables can be found in Todorov (2011) (Todorov et al., 2011, see also) and in Meraner (2010). Todorov et al. (2011) was written in a collaborative manner with UNIDO, the latter one, Meraner (2010), was written within the AMELI project where also details can be found in Hulliger et al. (2011b).

1.9.4 Work Package 8

Code for mapping and projection of coordinates are descibed in Deliverable 8.2 (Templ et al., 2011a).

Package `sparktable` (Kowarik et al., 2010) includes methods to generate scalable graphical tables including various types of sparklines for publication in web and in publications. It is mainly developed by Statistics Austria but with minor contribution from the AMELI team.

R code for checkerplots and further visualisation tools will be made soon available as a R package.

Bibliography

Alfons, A., **Burgard, J. P.**, **Filzmoser, P.**, **Hulliger, B.**, **Kolb, J.-P.**, **Kraft, S.**, **Münnich, R.**, **Schoch, T.** and **Templ, M.** (**2011**a): *The AMELI Simulation Study*. Research Project Report WP6 – D6.1, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net

Alfons, A., **Filzmoser, P.**, **Hulliger, B.**, **Kolb, J.-P.**, **Kraft, S.**, **Münnich, R.** and **Templ, M.** (**2011**b): *Synthetic Data Generation of SILC Data*. Research Project Report WP6 – D6.2, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net

Alfons, A., **Holzer, J.** and **Templ, M.** (**2011**c): laeken: Estimation of indicators on social exclusion and poverty. R package version 0.3.
URL http://CRAN.R-project.org/package=laeken

Alfons, A. and **Kraft, S.** (**2010**): simPopulation: Simulation of synthetic populations for surveys based on sample data. R package version 0.2.1.
URL http://CRAN.R-project.org/package=simPopulation

Alfons, A., **Kraft, S.**, **Templ, M.** and **Filzmoser, P.** (**2011**d): *Simulation of close-to-reality population data for household surveys with application to EU-SILC*. Statistical Methods & Applications, DOI 10.1007/s10260-011-0163-2, to appear.
URL http://dx.doi.org/10.1007/s10260-011-0163-2

Alfons, A., **Templ, M.** and **Filzmoser, P.** (**2010**a): *Applications of Statistical Simulation in the Case of EU-SILC: Using the R Package simFrame*. Journal of Statistical Software, 37 (3), p. 17, supplementary paper.
URL http://www.jstatsoft.org/v37/i03/

Alfons, A., **Templ, M.** and **Filzmoser, P.** (**2010**b): *An object-oriented framework for statistical simulation: The R package simFrame*. Journal of Statistical Software, 37 (3), pp. 1–36.
URL http://www.jstatsoft.org/v37/i03/

Chambers, J. (**1998**): Programming with Data. New York: Springer, ISBN 0-387-98503-4.

Chambers, J. (**2008**): Software for Data Analysis: Programming with R. New York: Springer, ISBN 978-0-387-75935-7.

Graf, M. and **Nedyalkova, D.** (**2010**): GB2: Generalized Beta Distribution of the Second Kind: properties, likelihood, estimation. R package version 1.0.
URL http://CRAN.R-project.org/package=GB2

Graf, M., **Nedyalkova, D.**, **Münnich, R.**, **Seger, J.** and **Zins, S.** (**2011**): *Parametric Estimation of Income Distributions and Indicators of Poverty and Social Exclusion*. Research Project Report WP2 – D2.1, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net

Hulliger, B., **Alfons, A.**, **Bruch, C.**, **Filzmoser, P.**, **Graf, M.**, **Kolb, J.-P.**, **Lehtonen, R.**, **Lussmann, D.**, **Meraner, A.**, **Münnich, R.**, **Nedyalkova, D.**, **Schoch, T.**, **Templ, M.**, **Valaste, M.**, **Veijanen, A.** and **Zins, S.** (**2011**a): *Report on the Simulation Results*. Research Project Report WP7 – D7.1, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net

Hulliger, B., **Alfons, A.**, **Filzmoser, P.**, **Meraner, A.**, **Schoch, T.** and **Templ, M.** (**2011**b): *Robust Methodology for Laeken Indicators*. Research Project Report WP4 – D4.2, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net

Kowarik, A., **Meindl, B.** and **Zechner, S.** (**2010**): sparkTable: Sparklines and graphical tables for tex and html. R package version 0.1.3.
URL http://CRAN.R-project.org/package=sparkTable

Lehtonen, R., **Veijanen, A.**, **Myrskylä, M.** and **Valaste, M.** (**2011**): *Small Area Estimation of Indicators on Poverty and Social Exclusion*. Research Project Report WP2 – D2.2, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net

Leisch, F. (**2003**): *Sweave, Part II: Package Vignettes*. R News, 3 (2), pp. 21–24.
URL http://CRAN.R-project.org/doc/Rnews/

Meraner, A. (**2010**): Outlier Detection for Semi-continuous Variables. Diplomarbeit, Institut f. Statistik und Wahrscheinlichkeitstheorie, Technische Universität, Wien.

Templ, M., **Alfons, A.**, **Filzmoser, P.**, **Hulliger, B.** and **Lussmann, D.** (**2011**a): *Visualisation Tools*. Research Project Report WP8 – D8.2, FP7-SSH-2007-217322 AMELI.
URL http://ameli.surveystatistics.net

Templ, M., **Alfons, A.** and **Kowarik, A.** (**2011**b): VIM: Visualization and Imputation of Missing Values. R package version 2.0.1.
URL http://CRAN.R-project.org/package=VIM

Templ, M., **Kowarik, A.** and **Filzmoser, P.** (**2011**c): *Iterative stepwise regression imputation using standard and robust methods*. Computational Statistics & Data Analysis, 55 (10), pp. 2793 – 2806, ISSN 0167-9473, doi:DOI:10.1016/j.csda.2011.04.012.
URL http://www.sciencedirect.com/science/article/pii/S0167947311001411

Tille, Y. and **Matei, A.** (**2011**): `sampling`: Survey Sampling. R package version 2.4.
URL http://CRAN.R-project.org/package=sampling

Todorov, V. (**2011**): rrcovNA: Scalable Robust Estimators with High Breakdown Point for Incomplete Data. R package version 0.4-02.
URL http://CRAN.R-project.org/package=rrcovNA

Todorov, V., **Templ, M.** and **Filzmoser, P.** (**2011**): *Detection of multivariate outliers in business survey data with incomplete information.* Advances in Data Analysis and Classification, 5, pp. 37–56, ISSN 1862-5347, doi:10.1007/s11634-010-0075-2.
URL http://dx.doi.org/10.1007/s11634-010-0075-2

Chapter 2

Applications of Statistical Simulation Using `simFrame`

Abstract: This chapter demonstrates the use of `simFrame` for various simulation designs in a practical application with EU-SILC data. It presents the full functionality of the framework regarding sampling designs, contamination models, missing data mechanisms and performing simulations separately on different domains. Due to the use of control objects, switching from one simulation design to another requires only minimal changes in the code. Using bespoke R code, on the other hand, changing the code to switch between simulation designs would require much greater effort. Furthermore, parallel computing with `simFrame` is demonstrated.

Keywords: R, statistical simulation, EU-SILC

2.1 Introduction

This is a supplementary paper to "An Object-Oriented Framework for Statistical Simulation: The R Package `simFrame`" (Alfons et al., 2010d) and demonstrates the use of `simFrame` (Alfons, 2011) in R (R Development Core Team, 2010) for various simulation designs in a practical application. It extends the example for design-based simulation in Alfons et al. (2010d) (Example 6.1). Different simulation designs in terms of sampling, contamination and missing data are thereby investigated to present the strengths of the framework.

Note that the paper is supplementary material and is supposed to be read after studying Alfons et al. (2010d). It does not give a detailed discussion about the motivation for the framework, nor does it describe the design or implementation of the package. Instead it is focused on showing its full functionality for design-based simulation in additional code examples with brief explanations. However, model-based simulation is not considered here.

The European Union Statistics on Income and Living Conditions (EU-SILC) is panel survey conducted in EU member states and other European countries and serves as basis for measuring risk-of-poverty and social cohesion in Europe. An important indicator calculated from this survey is the *Gini coefficient*, which is a well-known measure of

inequality. In the following examples, the standard estimation method (EU-SILC, 2004) is compared to two semiparametric methods under different simulation designs. The two semiparametric approaches are based on fitting a Pareto distribution (e.g., Kleiber and Kotz, 2003) to the upper tail of the data. In the first approach, the classical Hill estimator (Hill, 1975) is used to estimate the shape parameter of the Pareto distribution, while the second uses the robust partial density component (PDC) estimator (Vandewalle et al., 2007). All these methods are implemented in the R package `laeken` (Alfons et al., 2010a). For a more detailed discussion on Pareto tail modeling in the case of the Gini coefficient and a related measure of inequality, the reader is referred to Alfons et al. (2010e).

The example data set of `simFrame` is used as population data throughout the paper. It consists of 58 654 observations from 25 000 households and was synthetically generated from Austrian EU-SILC survey data from 2006 using the data simulation methodology by Alfons et al. (2010b), which is implemented R package `simPopulation` (Alfons and Kraft, 2010).

2.2 Application of different simulation designs to EU-SILC

First, the required packages and the data set need to be loaded.

```
R> library("simFrame")
R> library("laeken")
R> data("eusilcP")
```

Then, the function to be run in every iteration is defined. Its argument `k` determines the number of households whose income is modeled by a Pareto distribution. Since the Gini coefficient is calculated based on an equivalized household income, all individuals of a household in the upper tail receive the same value.

```
R> sim <- function(x, k) {
+     x <- x[!is.na(x$eqIncome), ]
+     g <- gini(x$eqIncome, x$.weight)$value
+     eqIncHill <- fitPareto(x$eqIncome, k = k, method = "thetaHill",
+         groups = x$hid)
+     gHill <- gini(eqIncHill, x$.weight)$value
+     eqIncPDC <- fitPareto(x$eqIncome, k = k, method = "thetaPDC",
+         groups = x$hid)
+     gPDC <- gini(eqIncPDC, x$.weight)$value
+     c(standard = g, Hill = gHill, PDC = gPDC)
+ }
```

This function is used in the following examples, which are designed to exhibit the strengths of the framework. In order to change from one simulation design to another, all there is to do is to define or modify control objects and supply them to the function `runSimulation()`.

2.2.1 Basic simulation design

In this basic simulation design, 100 samples of 1500 households are drawn using simple random sampling. Note that the `setup()` function is not used to permanently store the samples in an object. This is simply not necessary, since the population is rather small and the sampling method is straightforward. Furthermore, the Pareto distribution is fitted to the 175 households with the largest equivalized income.

```
R> set.seed(12345)
R> sc <- SampleControl(grouping = "hid", size = 1500, k = 100)
R> results <- runSimulation(eusilcP, sc, fun = sim, k = 175)
```

In order to inspect the simulation results, methods for several frequently used generic functions are implemented. Besides `head()`, `tail()` and `summary()` methods, a method for computing summary statistics with `aggregate()` is available. By default, the mean is used as summary statistic. Moreover, the `plot()` method selects a suitable graphical representation of the simulation results automatically. A reference line for the true value can thereby be added as well.

```
R> head(results)
```

```
  Run Sample standard     Hill      PDC
1   1      1 26.56793 26.48025 25.66614
2   2      2 26.98203 27.73124 26.39318
3   3      3 27.07081 27.11886 25.52524
4   4      4 26.86841 27.70216 25.71355
5   5      5 26.43215 26.49267 25.64191
6   6      6 26.96175 27.13876 27.17536
```

```
R> aggregate(results)
```

```
standard     Hill      PDC
26.65621 26.79016 26.89564
```

```
R> tv <- gini(eusilcP$eqIncome)$value
R> plot(results, true = tv)
```

Figure 2.1 shows the resulting box plots of the simulation results for the basic simulation design. While the PDC estimator comes with larger variability, all three methods are on average quite close to the true population value. This is also an indication that the choice of the number of households for fitting the Pareto distribution is suitable.

2.2.2 Using stratified sampling

The most frequently used sampling designs in official statistics are implemented in `simFrame`. In order to switch to another sampling design, only the corresponding control object needs to be changed. In this example, stratified sampling by region is performed. The sample sizes for the different strata are specified by using a vector for the slot `size` of the control object.

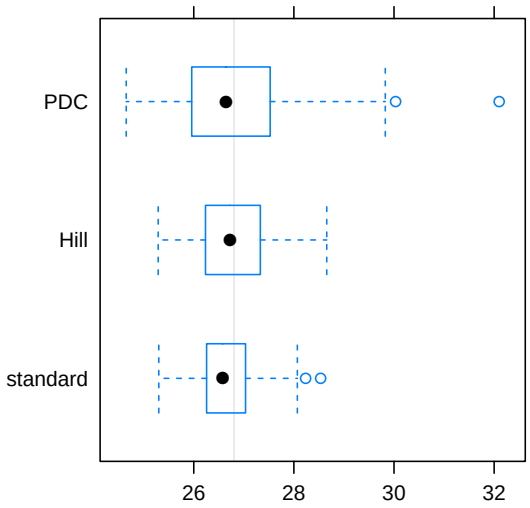

Figure 2.1: Simulation results for the basic simulation design.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100),
+     k = 100)
R> results <- runSimulation(eusilcP, sc, fun = sim, k = 175)
```

As before, the simulation results are inspected with `head()` and `aggregate()`. A plot of the simulation results is produced as well.

```
R> head(results)
```

```
  Run Sample standard     Hill      PDC
1   1      1 27.08652 27.22293 27.66753
2   2      2 26.80670 27.35874 25.93378
3   3      3 26.68113 27.03964 26.60062
4   4      4 25.84734 26.52346 25.18298
5   5      5 26.05449 26.26848 26.60331
6   6      6 26.98439 27.01396 26.48090
```

```
R> aggregate(results)
```

```
standard     Hill      PDC
26.71792 26.85375 26.86248
```

```
R> tv <- gini(eusilcP$eqIncome)$value
R> plot(results, true = tv)
```

Figure 2.2 contains the plot of the simulation results for the simulation design with stratified sampling. The results are very similar to those from the basic simulation design with simple random sampling. On average, all three investigated methods are quite close to the true population value.

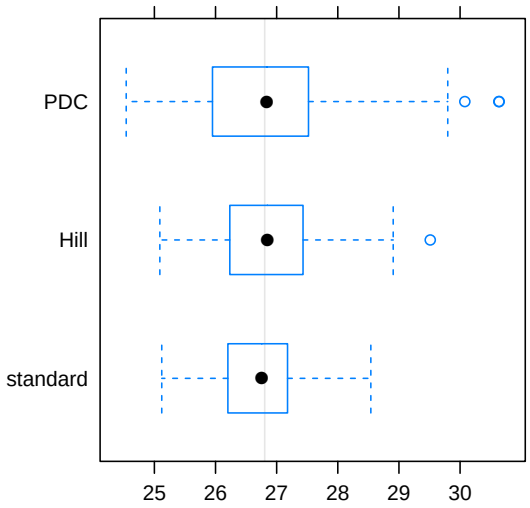

Figure 2.2: Simulation results for the simulation design with stratified sampling.

2.2.3 Adding contamination

When evaluating robust methods in simulation studies, contamination needs to be added to the data to study the influence of these outliers on the robust estimators and their classical counterparts. In `simFrame`, contamination is specified by defining a control object. Various contamination models are thereby implemented in the framework. Keep in mind that the term *contamination* is used in a technical sense here (see Alfons et al., 2010d,c, for an exact definition) and that contamination is modeled as a two step process (see also Béguin and Hulliger, 2008; Hulliger and Schoch, 2009). In this example, 0.5% of the households are selected to be contaminated using simple random sampling. The equivalized income of the selected households is then drawn from a normal distribution with mean $\mu = 500\,000$ and standard deviation $\sigma = 10\,000$.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100),
+     k = 100)
R> cc <- DCARContControl(target = "eqIncome", epsilon = 0.005,
+     grouping = "hid", dots = list(mean = 5e+05, sd = 10000))
R> results <- runSimulation(eusilcP, sc, contControl = cc,
+     fun = sim, k = 175)
```

The `head()`, `aggregate()` and `plot()` methods are again used to take a look at the simulation results. Note that a column is added that indicates the contamination level used.

```
R> head(results)
```

```
  Run Sample Epsilon standard     Hill      PDC
1   1      1   0.005 32.71453 29.12110 27.03731
```

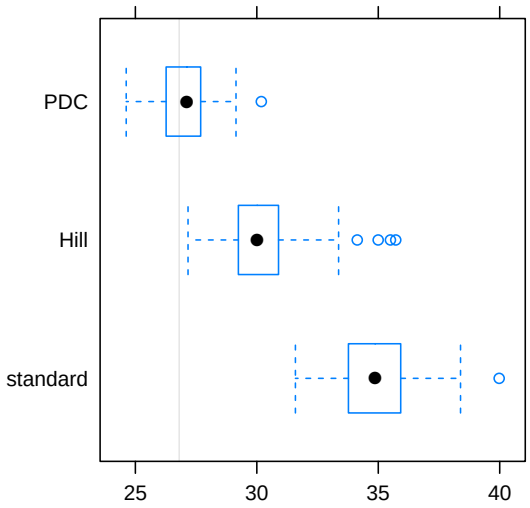

Figure 2.3: Simulation results for the simulation design with stratified sampling and contamination.

```
2   2       2   0.005 34.22065 31.62709 26.24857
3   3       3   0.005 33.56878 28.49760 28.00937
4   4       4   0.005 35.26346 29.57160 26.25621
5   5       5   0.005 33.79720 29.15945 25.61514
6   6       6   0.005 34.72069 28.58610 27.22342
```

```
R> aggregate(results)

  Epsilon standard     Hill      PDC
1   0.005 34.88922 30.26179 27.02093
```

```
R> tv <- gini(eusilcP$eqIncome)$value
R> plot(results, true = tv)
```

In Figure 2.3, the resulting box plots are presented. The figure shows that such a small amount of contamination is enough to completely corrupt the standard estimation of the Gini coefficient. Using the classical Hill estimator to fit the Pareto distribution is still highly influenced by the outliers, whereas the PDC estimator leads to very accurate results.

2.2.4 Performing simulations separately on different domains

Data sets from official statistics typically contain strong heterogeneities, therefore indicators are usually computed for subsets of the data as well. Hence it is often of interest to investigate the behavior of indicators on different subsets in simulation studies. In `simFrame`, this can be done by simply specifying the `design` argument of the function `runSimulation()`. In the case of extending the example from the previous section, the

framework then splits the samples, inserts contamination into each subset and calls the supplied function for these subsets automatically. With bespoke R code, the user would need to take care of this with a loop-like structure such as a `for` loop or a function from the `apply` family.

In the following example, the simulations are performed separately for each gender. It should be noted that the value of `k` for the Pareto distribution is thus changed to 125. This is the same as Example 6.1 from Alfons et al. (2010d), except that a control object for sampling is supplied to `runSimulation()` instead of setting up the samples beforehand and storing them in an object.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100),
+     k = 100)
R> cc <- DCARContControl(target = "eqIncome", epsilon = 0.005,
+     grouping = "hid", dots = list(mean = 5e+05, sd = 10000))
R> results <- runSimulation(eusilcP, sc, contControl = cc,
+     design = "gender", fun = sim, k = 125)
```

Below, the results are inspected using `head()` and `aggregate()`. The `aggregate()` method thereby computes the summary statistic for each subset automatically. Also the `plot()` method displays the results for the different subsets in different panels by taking advantage of the `lattice` system (Sarkar, 2008, 2010). In order to compute the true values for each subset, the function `simSapply()` is used.

```
R> head(results)
```

```
  Run Sample Epsilon gender standard     Hill      PDC
1   1      1   0.005   male 34.58446 29.96658 26.61415
2   1      1   0.005 female 38.82356 33.93700 28.82045
3   2      2   0.005   male 34.34853 29.09325 27.66380
4   2      2   0.005 female 36.38429 30.06097 27.42663
5   3      3   0.005   male 33.39992 30.54211 23.96698
6   3      3   0.005 female 35.12883 30.51336 26.06518
```

```
R> aggregate(results)
```

```
  Epsilon gender standard     Hill      PDC
1   0.005   male 33.18580 29.00265 26.21119
2   0.005 female 35.61341 31.28984 27.69054
```

```
R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

The resulting plots are shown in Figure 2.4, which is the same as Figure 2 in Alfons et al. (2010d). Clearly, the PDC estimator leads to excellent results for both subsets, while the two classical approaches are in both cases highly influenced by the outliers.

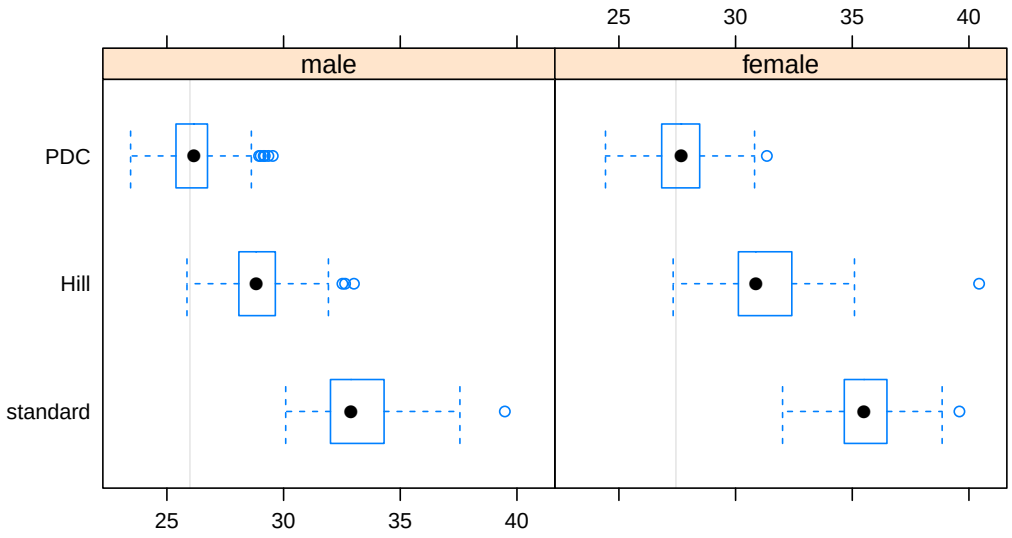

Figure 2.4: Simulation results for the simulation design with stratified sampling, contamination and performing the simulations separately for each gender.

2.2.5 Using multiple contamination levels

To get a more complete picture of the behavior of robust methods, more than one level of contamination is typically investigated in simulation studies. The only necessary modification of the code is to use a vector of contamination levels as the slot `epsilon` of the contamination control object. In this example, the contamination level is varied from 0% to 1% in steps of 0.25%. With bespoke R code, the user would have to add another loop-like structure to the code and collect the results in a suitable data structure. In `simFrame`, this is handled internally by the framework.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100),
+     k = 100)
R> cc <- DCARContControl(target = "eqIncome", epsilon = c(0,
+     0.0025, 0.005, 0.0075, 0.01), dots = list(mean = 5e+05,
+     sd = 10000))
R> results <- runSimulation(eusilcP, sc, contControl = cc,
+     design = "gender", fun = sim, k = 125)
```

The simulation results are inspected as usual. Note that the `aggregate()` method in this case returns values for each combination of contamination level and gender.

```
R> head(results)
```

```
  Run Sample Epsilon gender standard     Hill      PDC
1   1      1  0.0000   male 26.58067 26.50425 26.35969
2   1      1  0.0000 female 27.43355 27.03526 28.16992
```

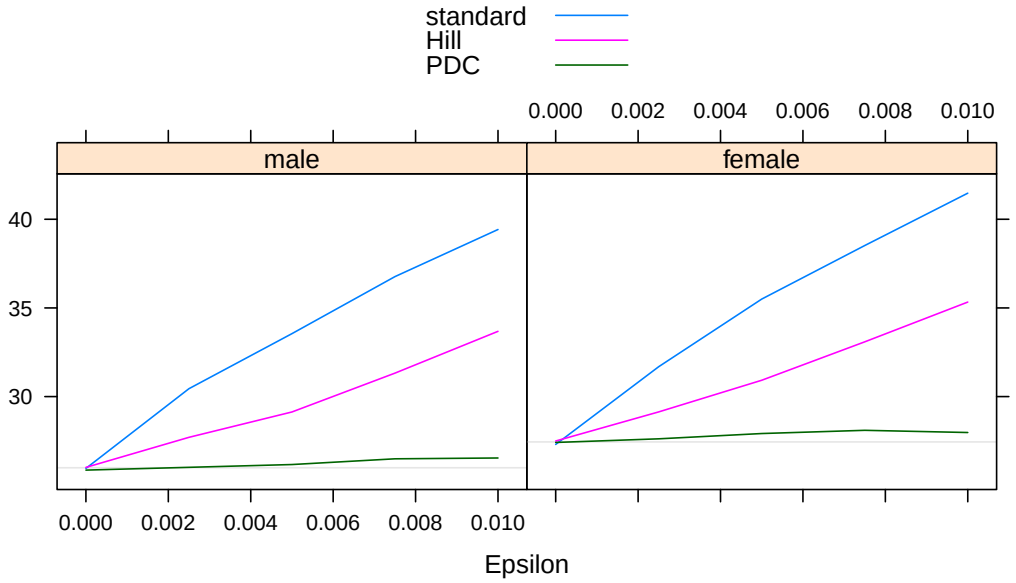

Figure 2.5: Simulation results for the simulation design with stratified sampling, multiple contamination levels and performing the simulations separately for each gender.

```
3   2      1  0.0025   male 31.63593 29.23365 27.12430
4   2      1  0.0025 female 31.43540 27.77698 26.85896
5   3      1  0.0050   male 33.35950 31.07040 25.97415
6   3      1  0.0050 female 35.68710 34.03560 29.11359
```

```
R> aggregate(results)
```

```
   Epsilon gender standard     Hill      PDC
1   0.0000   male 25.94937 26.00769 25.85311
2   0.0025   male 30.44448 27.70155 26.01033
3   0.0050   male 33.54929 29.13202 26.16786
4   0.0075   male 36.76641 31.32342 26.49026
5   0.0100   male 39.42281 33.67944 26.53749
6   0.0000 female 27.30171 27.49442 27.41323
7   0.0025 female 31.68505 29.13643 27.61790
8   0.0050 female 35.49976 30.92128 27.91607
9   0.0075 female 38.51819 33.08778 28.09784
10  0.0100 female 41.47137 35.32935 27.97407
```

```
R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

If multiple contamination levels are used in a simulation study, the `plot()` method for the simulation results no longer produces box plots. Instead, the average results are plotted against the corresponding contamination levels, as shown in Figure 2.5. The plots show how the classical estimators move away from the references line as the contamination level increases, while the values obtained with the PDC estimator remain quite accurate.

2.2.6 Inserting missing values

Survey data almost always contain a considerable amount of missing values. In close-to-reality simulation studies, the variability due to missing data therefore needs to be considered. Three types of missing data mechanisms are commonly distinguished in the literature (e.g., Little and Rubin, 2002): missing completely at random (MCAR), missing at random (MAR) and missing not at random (MNAR). All three missing data mechanisms are implemented in the framework.

In the following example, missing values are inserted into the equivalized household income of non-contaminated households with MCAR, i.e., the households whose values are going to be set to `NA` are selected using simple random sampling. In order to compare the scenario without missing values to a scenario with missing values, the missing value rates 0% and 5% are used. In the latter case, the missing values are simply disregarded for fitting the Pareto distribution and estimating the Gini coefficient. Furthermore, the number of samples is reduced to 50 and only the contamination levels 0%, 0.5% and 1% are investigated to keep the computation time of this motivational example low.

With `simFrame`, only a control object for missing data needs to be defined and supplied to `runSimulation()`, the rest is done automatically by the framework. To apply these changes to a simulation study implemented with bespoke R code, yet another loop-like structure for the different missing value rates as well as changes in the data structure for the simulation results would be necessary.

```
R> set.seed(12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100),
+     k = 50)
R> cc <- DCARContControl(target = "eqIncome", epsilon = c(0,
+     0.005, 0.01), dots = list(mean = 5e+05, sd = 10000))
R> nc <- NAControl(target = "eqIncome", NArate = c(0, 0.05))
R> results <- runSimulation(eusilcP, sc, contControl = cc,
+     NAControl = nc, design = "gender", fun = sim, k = 125)
```

As always, the `head()`, `aggregate()` and `plot()` methods are used to take a look at the simulation results. It should be noted that a column is added to the results that indicates the missing value rate used and that `aggregate()` in this example returns a value for each combination of contamination level, missing value rate and gender.

```
R> head(results)
```

```
  Run Sample Epsilon NArate gender standard     Hill      PDC
1   1      1   0.000   0.00   male 26.58067 27.00998 26.26273
2   1      1   0.000   0.00 female 27.43355 27.92305 26.69034
3   2      1   0.000   0.05   male 26.62313 26.54198 26.01043
4   2      1   0.000   0.05 female 27.51209 26.83574 27.25464
5   3      1   0.005   0.00   male 33.71363 28.44824 26.46635
6   3      1   0.005   0.00 female 35.47508 28.48208 27.70783
```

```
R> aggregate(results)
```

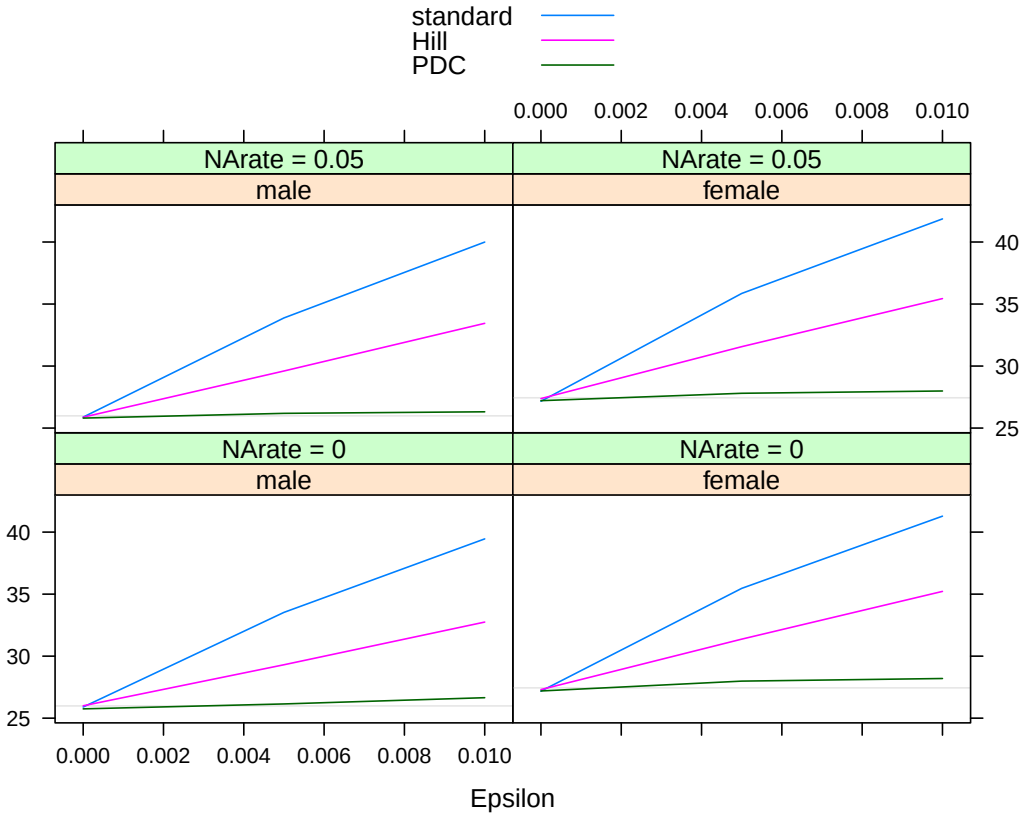

Figure 2.6: Simulation results for the simulation design with stratified sampling, multiple contamination levels, multiple missing value rates and performing the simulations separately for each gender.

```
   Epsilon NArate gender standard     Hill      PDC
1    0.000    0.00   male 25.89948 25.99777 25.74944
2    0.005    0.00   male 33.52791 29.30477 26.14659
3    0.010    0.00   male 39.45422 32.74672 26.64929
4    0.000    0.05   male 25.88434 25.87824 25.80541
5    0.005    0.05   male 33.87975 29.60079 26.18759
6    0.010    0.05   male 39.99526 33.44462 26.31274
7    0.000    0.00 female 27.17769 27.30586 27.19275
8    0.005    0.00 female 35.46414 31.37099 27.98622
9    0.010    0.00 female 41.28625 35.22113 28.19677
10   0.000    0.05 female 27.16026 27.37710 27.20892
11   0.005    0.05 female 35.85305 31.56317 27.80455
12   0.010    0.05 female 41.86453 35.44025 27.98948
```

```
R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

If multiple contamination levels and multiple missing value rates are used in the simulation study, conditional plots are produced by the `plot()` method for the simulation results. Figure 2.6 shows the resulting plots for this example. The bottom panels illustrate the

scenario without missing values, while the scenario with 5% missing values is displayed in the top panels. In this case, there is not much of a difference in the results for the two scenarios.

2.2.7 Parallel computing

Statistical simulation is an *embarrassingly parallel* procedure, hence parallel computing can drastically reduce the computational costs. In `simFrame`, parallel computing is implemented using `snow` (Rossini et al., 2007; Tierney et al., 2008). Only minimal additional programming effort due to the use of `snow` is required to adapt the code from the previous example: to initialize the computer cluster, to ensure that all packages and objects are available on each worker process, to use the function `clusterRunSimulation()` instead of `runSimulation()` and to stop the computer cluster after the simulations. In addition, random number streams (e.g., L'Ecuyer et al., 2002; Sevcikova and Rossini, 2009) should be used instead of the built-in random number generator.

```
R> cl <- makeCluster(4, type = "SOCK")
R> clusterEvalQ(cl, {
+     library("simFrame")
+     library("laeken")
+     data("eusilcP")
+ })
R> clusterSetupRNG(cl, seed = 12345)
R> sc <- SampleControl(design = "region", grouping = "hid",
+     size = c(75, 250, 250, 125, 200, 225, 125, 150, 100),
+     k = 50)
R> cc <- DCARContControl(target = "eqIncome", epsilon = c(0,
+     0.005, 0.01), dots = list(mean = 5e+05, sd = 10000))
R> nc <- NAControl(target = "eqIncome", NArate = c(0, 0.05))
R> clusterExport(cl, c("sc", "cc", "nc", "sim"))
R> results <- clusterRunSimulation(cl, eusilcP, sc, contControl = cc,
+     NAControl = nc, design = "gender", fun = sim, k = 125)
R> stopCluster(cl)
```

When the parallel computations are finished and the simulation results are obtained, they can be inspected as usual.

```
R> head(results)
```

```
  Run Sample Epsilon NArate gender standard     Hill      PDC
1   1      1   0.000   0.00   male 26.20067 27.02017 23.66565
2   1      1   0.000   0.00 female 28.79194 29.23548 27.12933
3   2      1   0.000   0.05   male 26.19328 24.91570 24.07906
4   2      1   0.000   0.05 female 28.86860 27.38585 27.80012
5   3      1   0.005   0.00   male 34.46084 31.74470 24.87023
6   3      1   0.005   0.00 female 36.27429 32.14269 28.06137
```

```
R> aggregate(results)
```

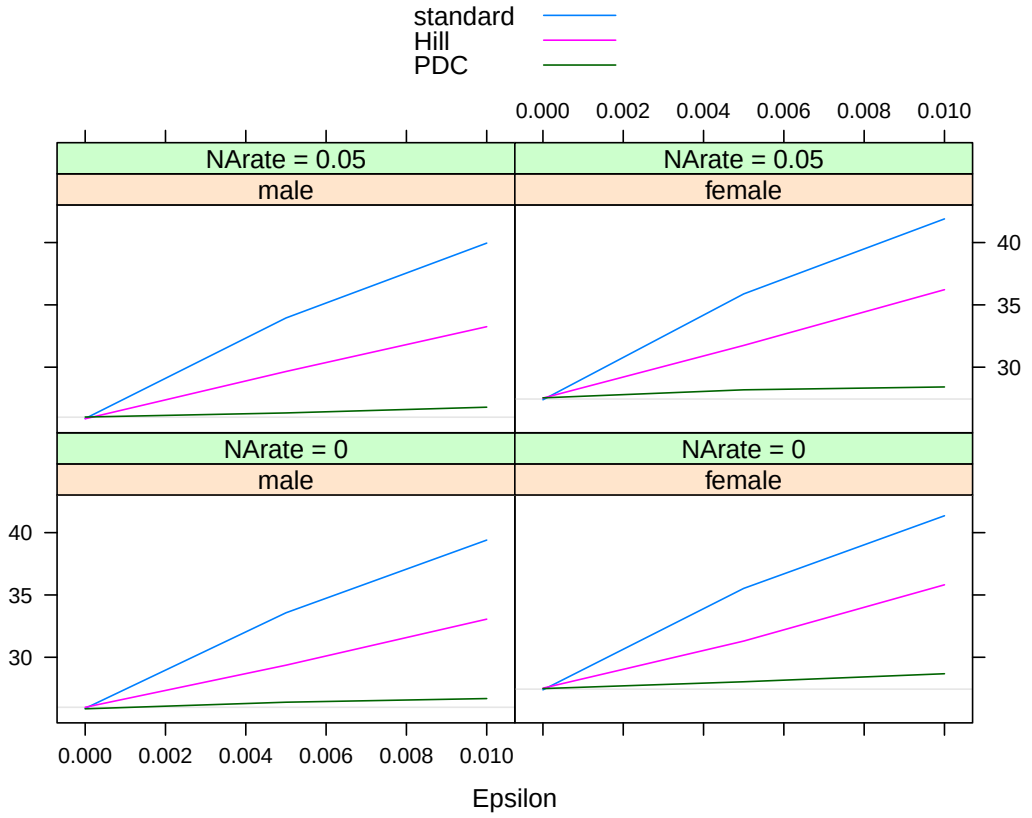

Figure 2.7: Simulation results obtained by parallel computing for the simulation design with stratified sampling, multiple contamination levels, multiple missing value rates and performing the simulations separately for each gender.

```
   Epsilon NArate gender standard     Hill      PDC
1    0.000    0.00   male 25.89996 25.98977 25.86451
2    0.005    0.00   male 33.56743 29.36361 26.39515
3    0.010    0.00   male 39.40362 33.05926 26.68715
4    0.000    0.05   male 25.87909 25.86055 26.00109
5    0.005    0.05   male 33.94829 29.65456 26.32813
6    0.010    0.05   male 39.95535 33.24853 26.78947
7    0.000    0.00 female 27.38636 27.52210 27.48816
8    0.005    0.00 female 35.52688 31.30099 28.03385
9    0.010    0.00 female 41.35311 35.81549 28.67901
10   0.000    0.05 female 27.38459 27.51825 27.54063
11   0.005    0.05 female 35.87991 31.74678 28.18308
12   0.010    0.05 female 41.89804 36.21921 28.41367
```

```
R> tv <- simSapply(eusilcP, "gender", function(x) gini(x$eqIncome)$value)
R> plot(results, true = tv)
```

Figure 4.9 shows the simulation results obtained with parallel computing. The plots are, of course, very similar to the plots for the previous example in Figure 2.6, since the design of the simulation studies is the same.

2.3 Conclusions

In this paper, the use of the R package `simFrame` for different simulation designs has been demonstrated in a practical application. The full functionality of the framework for design-based simulation has been presented in various code examples. These examples showed that the framework allows researchers to make use of a wide range of simulation designs with only a few lines of code. In order to switch from one simulation design to another, only control objects need to be defined or modified. Even moving from basic to highly complex designs therefore requires only minimal changes to the code. With bespoke R code, such modifications would often need a considerable amount of programming. Furthermore, parallel computing with `simFrame` can easily be done based on package `snow`.

Besides the functionality for carrying out simulation studies, methods for several frequently used generic functions are available for inspecting or summarizing the simulation results. Most notably, a suitable plot method of the simulation results is selected automatically depending on their structure.

Due to this flexibility, `simFrame` is widely applicable for gaining insight into the quality of statistical methods and is a valuable addition to a researcher's toolbox.

Bibliography

Alfons, A. (**2011**): `simFrame`: Simulation Framework. R package version 0.4.1.
URL http://CRAN.R-project.org/package=simFrame

Alfons, A., **Holzer, J.** and **Templ, M.** (**2010**a): `laeken`: Laeken Indicators for Measuring Social Cohesion. R package version 0.1.3.
URL http://CRAN.R-project.org/package=laeken

Alfons, A. and **Kraft, S.** (**2010**): `simPopulation`: Simulation of Synthetic Populations for Surveys based on Sample Data. R package version 0.2.
URL http://CRAN.R-project.org/package=simPopulation

Alfons, A., **Kraft, S.**, **Templ, M.** and **Filzmoser, P.** (**2010**b): *Simulation of Synthetic Population Data for Household Surveys with Application to EU-SILC*. Research Report CS-2010-1, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2010-1complete.pdf

Alfons, A., **Templ, M.** and **Filzmoser, P.** (**2010**c): *Contamination Models in the R Package `simFrame` for Statistical Simulation*. **Aivazian, S.**, **Filzmoser, P.** and **Kharin, Y.** (editors) Computer Data Analysis and Modeling: Complex Stochastic Data and Systems, vol. 2, pp. 178–181, Minsk, ISBN 978-985-476-848-9.

Alfons, A., **Templ, M.** and **Filzmoser, P.** (**2010**d): *An Object-Oriented Framework for Statistical Simulation: The R Package `simFrame`*. Journal of Statistical Software, 37 (3), pp. 1–36.
URL http://www.jstatsoft.org/v37/i03/

Alfons, A., **Templ, M.**, **Filzmoser, P.** and **Holzer, J.** (**2010**e): *A Comparison of Robust Methods for Pareto Tail Modeling in the Case of Laeken Indicators*. **Borgelt, C.**, **González-Rodríguez, G.**, **Trutschnig, W.**, **Lubiano, M.**, **Gil, M.**, **Grzegorzewski, P.** and **Hryniewicz, O.** (editors) Combining Soft Computing and Statistical Methods in Data Analysis, *Advances in Intelligent and Soft Computing*, vol. 77, pp. 17–24, Heidelberg: Springer-Verlag, ISBN 978-3-642-14745-6.

Béguin, C. and **Hulliger, B.** (**2008**): *The BACON-EEM Algorithm for Multivariate Outlier Detection in Incomplete Survey Data*. Survey Methodology, 34 (1), pp. 91–103.

EU-SILC (**2004**): *Common Cross-Sectional EU Indicators based on EU-SILC; the Gender Pay Gap*. EU-SILC 131-rev/04, Working group on Statistics on Income and Living Conditions (EU-SILC), Eurostat, Luxembourg.

Hill, B. (**1975**): *A Simple General Approach to Inference about the Tail of a Distribution*. The Annals of Statistics, 3 (5), pp. 1163–1174.

Hulliger, B. and **Schoch, T.** (**2009**): *Robust Multivariate Imputation with Survey Data*. 57th Session of the International Statistical Institute, Durban.

Kleiber, C. and **Kotz, S.** (**2003**): Statistical Size Distributions in Economics and Actuarial Sciences. Hoboken: John Wiley & Sons, ISBN 0-471-15064-9.

L'Ecuyer, P., **Simard, R.**, **Chen, E.** and **Kelton, W.** (**2002**): *An Object-Oriented Random-Number Package with Many Long Streams and Substreams*. Operations Research, 50 (6), pp. 1073–1075.

Little, R. and **Rubin, D.** (**2002**): Statistical Analysis with Missing Data. New York: John Wiley & Sons, 2nd ed., ISBN 0-471-18386-5.

R Development Core Team (**2010**): R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Rossini, A., **Tierney, L.** and **Li, N.** (**2007**): *Simple Parallel Statistical Computing in R*. Journal of Computational and Graphical Statistics, 16 (2), pp. 399–420.

Sarkar, D. (**2008**): Lattice: Multivariate Data Visualization with R. New York: Springer-Verlag, ISBN 978-0-387-75968-5.

Sarkar, D. (**2010**): `lattice`: Lattice Graphics. R package version 0.19-13.
URL http://CRAN.R-project.org/package=lattice

Sevcikova, H. and **Rossini, T.** (**2009**): `rlecuyer`: R Interface to RNG with Multiple Streams. R package version 0.3-1.
URL http://CRAN.R-project.org/package=rlecuyer

Tierney, L., **Rossini, A.**, **Li, N.** and **Sevcikova, H.** (**2008**): `snow`: Simple Network of Workstations. R package version 0.3-3.
URL http://CRAN.R-project.org/package=snow

Vandewalle, B., **Beirlant, J.**, **Christmann, A.** and **Hubert, M.** (**2007**): *A Robust Estimator for the Tail Index of Pareto-Type Distributions*. Computational Statistics & Data Analysis, 51 (12), pp. 6252–6268.

Chapter 3

Simulation of EU-SILC Population Data Using `simPopulation`

Abstract: This vignette demonstrates the use of `simPopulation` for simulating population data in an application to the EU-SILC example data from the package. It presents a wrapper function tailored specifically towards EU-SILC data for convenience and ease of use, as well as detailed instructions for performing each of the four involved data generation steps separately. In addition, the generation of diagnostic plots for the simulated population data is illustrated.

Keywords: R, synthetic data, simulation, survey statistics, EU-SILC

3.1 Introduction

This package vignette is a companion to Alfons et al. (2010) that shows how the proposed framework for the simulation of population data can be applied in R (R Development Core Team, 2010) using the package `simPopulation` (Alfons and Kraft, 2010). The data simulation framework consists of four steps:

1. Setup of the household structure
2. Simulation of categorical variables
3. Simulation of (semi-)continuous variables
4. Splitting (semi-)continuous variables into components

Note that this vignette does not motivate, describe or evaluate the statistical methodology of the framework. Instead it is focused on the R code to generate synthetic population data and produce diagnostic plots. For details on the statistical methodology, the reader is referred to Alfons et al. (2010).

The *European Union Statistics on Income and Living Conditions* (EU-SILC) is panel survey conducted in European countries and serves as data basis for the estimation social inclusion indicators in Europe. EU-SILC data are highly complex and contain detailed information on the income of the sampled individuals and households. More information on EU-SILC can be found in Eurostat (2004).

In Alfons et al. (2010), three methods for the simulation of the net income of the individuals in the population are proposed and analyzed:

MP Multinomial logistic regression models with random draws from the resulting categories. For the categories corresponding to the upper tail, the values are drawn from a (truncated) generalized Pareto distribution, for the other categories from a uniform distribution.

TR Two-step regression models with trimming and random draws from the residuals.

TN Two-step regression models with trimming and random draws from a normal distribution.

The first two steps of the analysis, namely the simulation of the household structure and additional categorical variables, are performed in exactly the same manner for the three scenarios. While the simulation of the income components is carried out with the same parameter settings, the results of course depend on the simulated net income.

It is important to note that the original Austrian EU-SILC sample provided by Statistics Austria and used in Alfons et al. (2010) is confidential, hence the results presented there cannot be reproduced in this vignette. Nevertheless, the code for such an analysis is presented here using the example data from the package, which has been synthetically generated itself. In fact, this example data set is a sample drawn from one of the populations generated in Alfons et al. (2010). However, the sample weights have been modified such that the size of the resulting populations is about 1% of the real Austrian population in order to keep the computation time low. Table 3.1 lists the variables of the example data used in the code examples.

With the following commands, the package and the example data are loaded. Furthermore, the numeric value stored in `seed` will be used as seed for the random number generator in the examples to make the results reproducible.

```
R> library("simPopulation")
R> data("eusilcS")
R> seed <- 1234
```

The rest of this vignette is organized as follows. Section 3.2 illustrates the use of a convenient wrapper function for the generation of EU-SILC population data. In Section 3.3, detailed instructions are given for each step in the data generation process as well as for the generation of diagnostic plots. The final Section 3.4 concludes.

3.2 Wrapper function for EU-SILC

A convenient way of generating synthetic EU-SILC population data is provided by the wrapper function `simEUSILC()`, which performs the four steps of the data simulation procedure at once. For each step, the names of the variables to be simulated can be supplied. However, the default values for the respective arguments are given by the variables names used in Alfons et al. (2010). Since the same names are used in the example data, the complex procedures for the three different methods can be carried out with very simple commands.

Table 3.1: Variables of the EU-SILC example data in `simPopulation`.

Variable	Name	Type	
Region	`db040`	Categorical	9 levels
Household size	`hsize`	Categorical	9 levels
Age	`age`	Categorical	
Gender	`rb090`	Categorical	2 levels
Economic status	`pl030`	Categorical	7 levels
Citizenship	`pb220a`	Categorical	3 levels
Personal net income	`netIncome`	Semi-continuous	
Employee cash or near cash income	`py010n`	Semi-continuous	
Cash benefits or losses from self-employment	`py050n`	Semi-continuous	
Unemployment benefits	`py090n`	Semi-continuous	
Old-age benefits	`py100n`	Semi-continuous	
Survivor's benefits	`py110n`	Semi-continuous	
Sickness benefits	`py120n`	Semi-continuous	
Disability benefits	`py130n`	Semi-continuous	
Education-related allowances	`py140n`	Semi-continuous	
Household sample weights	`db090`	Continuous	
Personal sample weights	`rb050`	Continuous	

```
R> eusilcMP <- simEUSILC(eusilcS, upper = 2e+05, equidist = FALSE,
+     seed = seed)
R> eusilcTR <- simEUSILC(eusilcS, method = "twostep", seed = seed)
R> eusilcTN <- simEUSILC(eusilcS, method = "twostep", residuals = FALSE,
+     seed = seed)
```

Note that the default is to use the MP procedure. An upper bound for the net income is supplied using the argument `upper`, while the argument `equidist` is set to `FALSE` so that the breakpoints for the discretization of the net income are given by quantiles with non-equidistant probabilities as described in Alfons et al. (2010). The twostep regression approaches are performed by setting `method = 'twostep'`, in which case the logical argument `residuals` specifies whether variability should be added by random draws from the residuals (TR method, the default) or from a normal distribution (TN method). In both cases, the default trimming parameter `alpha = 0.01` is used.

The synthetic populations generated with the wrapper function are not further evaluated here, instead a detailed illustration of each step along with diagnostic plots is provided in the following section.

3.3 Step by step instructions and diagnostics

As for the wrapper function `simEUSILC()`, the variable names of the example data set are used as default values for the corresponding arguments of the functions for the different steps of the procedure. Nevertheless, in order to demonstrate how these arguments are

used, the names of the involved variables are always supplied in the commands shown in this section.

The first step of the analysis is to set up the basic household structure using the function `simStructure()`. Note that a variable named `'hsize'` giving the household sizes is generated automatically in this example, but the name of the corresponding variable in the sample data can also be specified as an argument. Furthermore, the argument `additional` specifies the variables that define the household structure in addition to the household size (in this case age and gender).

```
R> eusilcP <- simStructure(eusilcS, hid = "db030", w = "db090",
+     strata = "db040", additional = c("age", "rb090"))
```

For the rest of the procedure, combined age categories are used for the individuals in order to reduce the computation time of the statistical models.

```
R> breaks <- c(min(eusilcS$age), seq(15, 80, 5), max(eusilcS$age))
R> eusilcS$ageCat <- as.character(cut(eusilcS$age, breaks = breaks,
+     include.lowest = TRUE))
R> eusilcP$ageCat <- as.character(cut(eusilcP$age, breaks = breaks,
+     include.lowest = TRUE))
```

Additional categorical variables are then simulated using the function `simCategorical()`. The argument `basic` thereby specifies the already generated variables for the basic household structure (age category, gender and household size), while `additional` specifies the variables to be simulated in this step (economic status and citizenship).

```
R> basic <- c("ageCat", "rb090", "hsize")
R> eusilcP <- simCategorical(eusilcS, eusilcP, w = "rb050", strata = "db040",
+     basic = basic, additional = c("pl030", "pb220a"))
```

Mosaic plots are available as graphical diagnostic tools for checking whether the structures of categorical variables are reflected in the synthetic population. They are implemented in the function `spMosaic()` based on the package `vcd` (Meyer et al., 2006, 2010), which contains extensive functionality for customization.

With the following commands, mosaic plots for the variables gender, region and household size are created (see Figure 3.1, *top*). The function `labeling_border()` from package `vcd` is thereby used to set shorter labels for the different regions and to display more meaningful labels for the variables.

```
R> abb <- c("B", "LA", "Vi", "C", "St", "UA", "Sa", "T", "Vo")
R> nam <- c(rb090 = "Gender", db040 = "Region", hsize = "Household size")
R> lab <- labeling_border(set_labels = list(db040 = abb),
+     set_varnames = nam)
R> spMosaic(c("rb090", "db040", "hsize"), "rb050", eusilcS,
+     eusilcP, labeling = lab)
```

In addition, mosaic plots for the variables gender, economic status and citizenship are produced (see Figure 3.1, *bottom*). Also in this case, `labeling_border()` is used for some fine tuning. In particular, the categories of citizenship are abbreviated and again more meaningful labels for the variables are set.

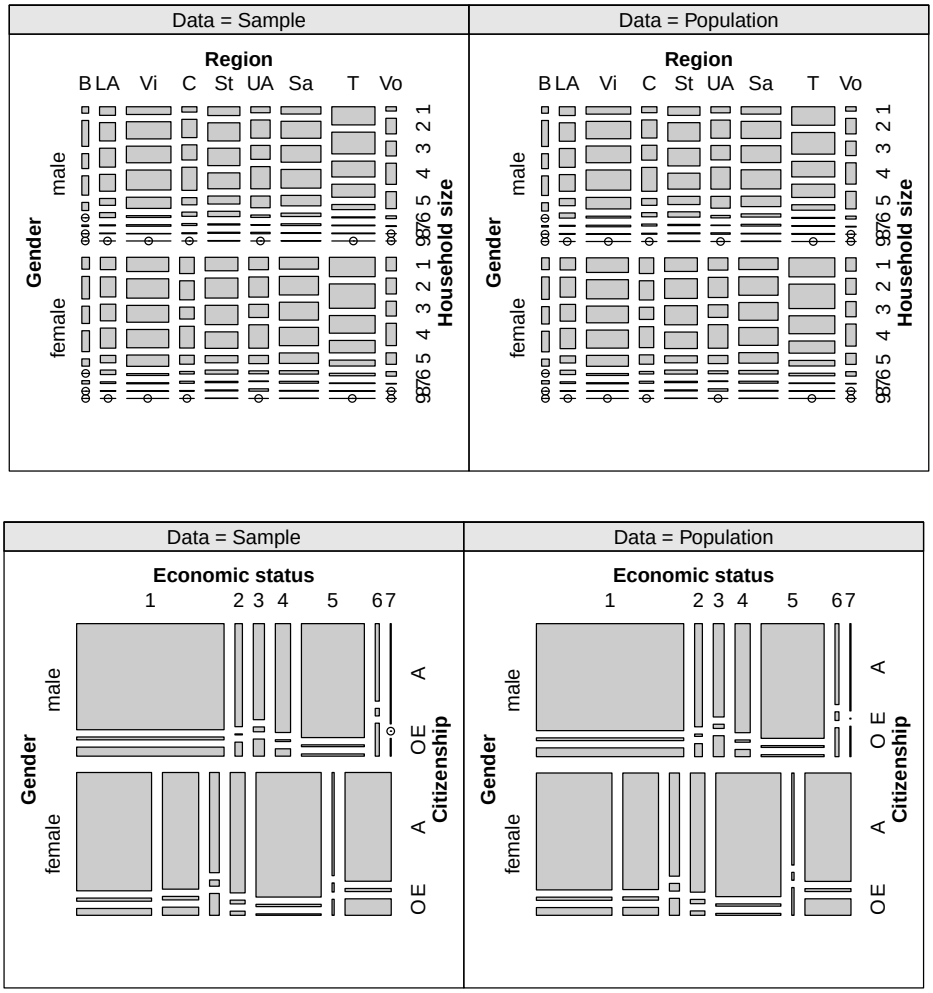

Figure 3.1: *Top*: Mosaic plots of gender, region and household size. *Bottom*: Mosaic plots of gender, economic status and citizenship.

```
R> nam <- c(rb090 = "Gender", pl030 = "Economic status",
+     pb220a = "Citizenship")
R> lab <- labeling_border(abbreviate = c(FALSE, FALSE, TRUE),
+     set_varnames = nam)
R> spMosaic(c("rb090", "pl030", "pb220a"), "rb050", eusilcS,
+     eusilcP, labeling = lab)
```

Next, the function `simContinuous()` is used to simulate the net income according to the three proposed methods. The same parameter settings as in Section 3.2 are thereby used for each of the methods. In any case, the argument `basic` specifies the predictor variables (age category, gender, household size, economic status and citizenship), while the argument `additional` specifies the variable to be simulated.

Note that the current state of the random number generator is stored beforehand so that the different methods can all be started with the same seed. Furthermore, the random seed after each of the methods has finished is stored so that the simulation of the income components can later on continue from there.

```
R> seedP <- .Random.seed
R> basic <- c(basic, "pl030", "pb220a")
R> eusilcMP <- simContinuous(eusilcS, eusilcP, w = "rb050",
+     strata = "db040", basic = basic, additional = "netIncome",
+     upper = 2e+05, equidist = FALSE, seed = seedP)
R> seedMP <- .Random.seed
R> eusilcTR <- simContinuous(eusilcS, eusilcP, w = "rb050",
+     strata = "db040", basic = basic, additional = "netIncome",
+     method = "lm", seed = seedP)
R> seedTR <- .Random.seed
R> eusilcTN <- simContinuous(eusilcS, eusilcP, w = "rb050",
+     strata = "db040", basic = basic, additional = "netIncome",
+     method = "lm", residuals = FALSE, seed = seedP)
R> seedTN <- .Random.seed
```

Two functions are available as diagnostic tools for (semi-)continuous variables: `spCdfplot()` for comparing the cumlative distribution functions, and `spBwplot()` for comparisons with box-and-whisker plots. Both are implemented based on the package `lattice` (Sarkar, 2008, 2010).

The following commands are used to produce the two plots in Figure 3.2. For better visibility of the differences in the main parts of the cumulative distribution functions, only the parts between 0 and the weighted 99% quantile of the sample are plotted (see Figure 3.2, *left*). Furthermore, the box-and-whisker plots by default do not display any points outside the extremes of the whiskers (see Figure 3.2, *right*). This is because population data are typically very large, which almost always would result in a large number of observations ouside the whiskers. Also note that a list containing the three populations is supplied as the argument `dataP` of the plot functions.

```
R> subset <- which(eusilcS[, "netIncome"] > 0)
R> q <- quantileWt(eusilcS[subset, "netIncome"], eusilcS[subset,
+     "rb050"], probs = 0.99)
R> listP <- list(MP = eusilcMP, TR = eusilcTR, TN = eusilcTN)
R> spCdfplot("netIncome", "rb050", dataS = eusilcS, dataP = listP,
+     xlim = c(0, q))
R> spBwplot("netIncome", "rb050", dataS = eusilcS, dataP = listP,
+     pch = "|")
```

One of the main requirements in the simulation of population data is that heterogeneities between subgroups are reflected (see Alfons et al., 2010). Since `spCdfplot()` and `spBwplot()` are based on `lattice`, this can easily be checked by producing conditional plots. With the following commands, the box-and-whisker plots in Figure 3.3 are produced. The conditioning variables gender (*top left*), citizenship (*top right*), region (*bottom left*) and economic status (*bottom right*) are thereby used. For finetuning, the layout of the panels is specified with the `layout` argument provided by the `lattice` framework.

```
R> spBwplot("netIncome", "rb050", "rb090", dataS = eusilcS,
+     dataP = listP, pch = "|", layout = c(1, 2))
R> spBwplot("netIncome", "rb050", "pb220a", dataS = eusilcS,
+     dataP = listP, pch = "|", layout = c(1, 3))
```

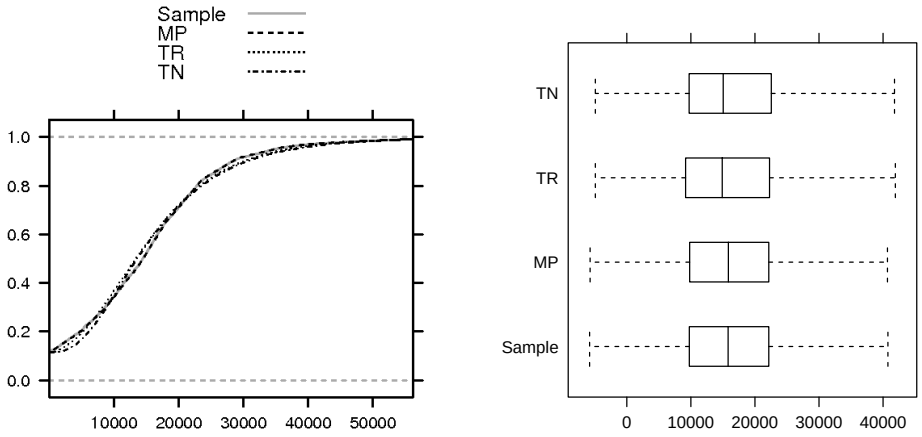

Figure 3.2: *Left*: Cumulative distribution functions of personal net income. For better visibility, the plot shows only the main parts of the data. *Right:* Box plots of personal net income. Points outside the extremes of the whiskers are not plotted.

```
R> spBwplot("netIncome", "rb050", "db040", dataS = eusilcS,
+     dataP = listP, pch = "|", layout = c(1, 9))
R> spBwplot("netIncome", "rb050", "pl030", dataS = eusilcS,
+     dataP = listP, pch = "|", layout = c(1, 7))
```

The last step of the analysis is to simulate the income components. This is done based on resampling of fractions conditional on net income category and economic status. Therefore, the net income categories need to be constructed first. With the function `getBreaks()`, default breakpoints based on quantiles are computed. In this example, the argument `upper` is set to `Inf` to avoid problems with different maximum values in the three synthetic populations, and the argument `equidist` is set to `FALSE` such that non-equidistant probabilities as described in Alfons et al. (2010) are used for the calculation of the quantiles.

```
R> breaks <- getBreaks(eusilcS$netIncome, eusilcS$rb050,
+     upper = Inf, equidist = FALSE)
R> eusilcS$netIncomeCat <- getCat(eusilcS$netIncome, breaks)
R> eusilcMP$netIncomeCat <- getCat(eusilcMP$netIncome, breaks)
R> eusilcTR$netIncomeCat <- getCat(eusilcTR$netIncome, breaks)
R> eusilcTN$netIncomeCat <- getCat(eusilcTN$netIncome, breaks)
```

Once the net income categories are constructed, the income components are simulated using the function `simComponents()`. The arguments `total`, `components` and `conditional` thereby specify the variable to be split, the variables containing the components, and the conditioning variables, respectively. In addition, for each of the three populations the seed of the random number generator is set to the corresponding state after the simulation of the net income.

```
R> components <- c("py010n", "py050n", "py090n", "py100n",
+     "py110n", "py120n", "py130n", "py140n")
```

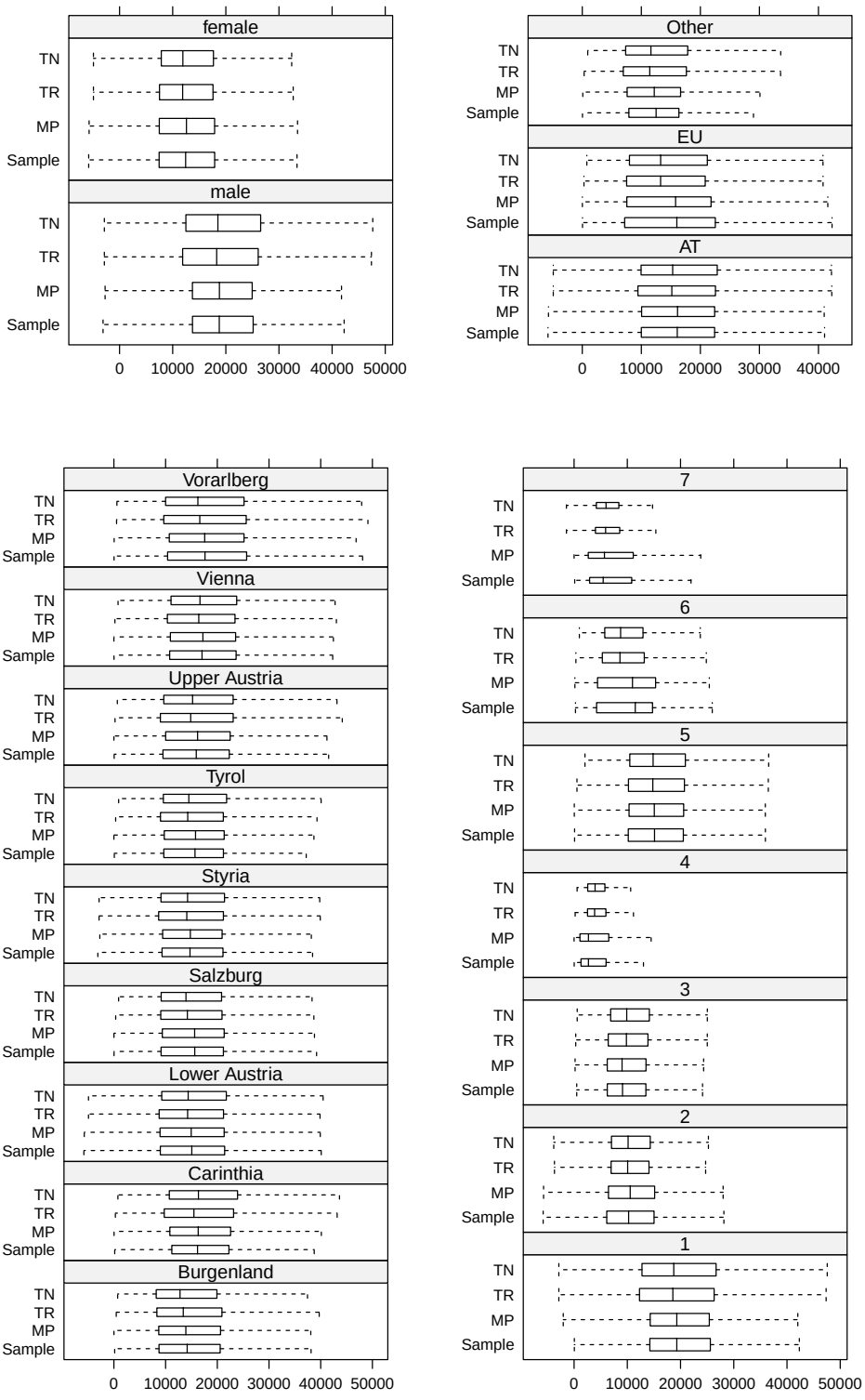

Figure 3.3: Box plots of personal net income split by gender (*top left*), citizenship (*top right*), region (*bottom left*) and economic status (*bottom right*). Points outside the extremes of the whiskers are not plotted.

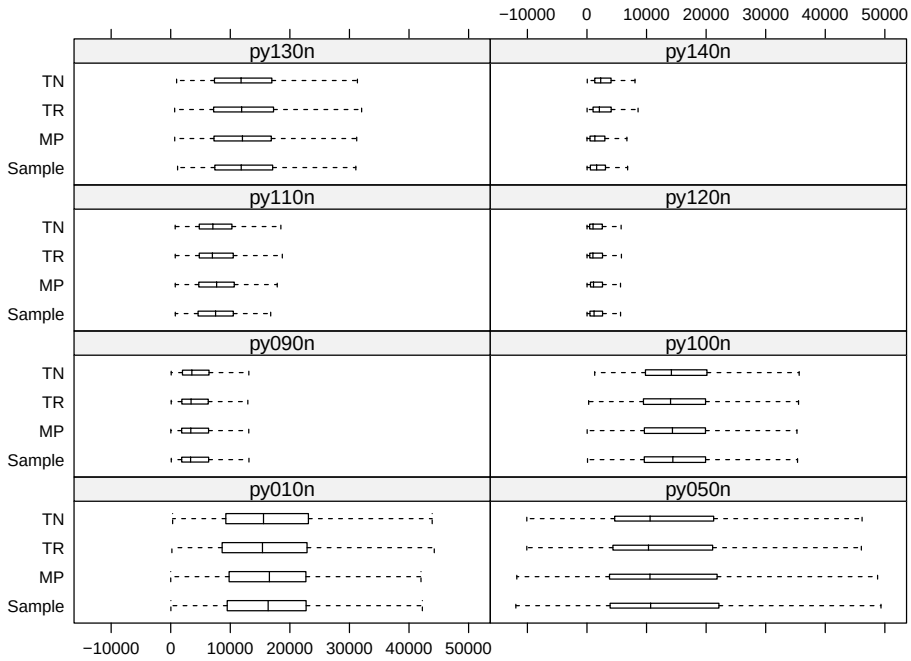

Figure 3.4: Box plots of the income components. Points outside the extremes of the whiskers are not plotted.

```
R> eusilcMP <- simComponents(eusilcS, eusilcMP, w = "rb050",
+     total = "netIncome", components = components,
+     conditional = c("netIncomeCat", "pl030"), seed = seedMP)
R> eusilcTR <- simComponents(eusilcS, eusilcTR, w = "rb050",
+     total = "netIncome", components = components,
+     conditional = c("netIncomeCat", "pl030"), seed = seedTR)
R> eusilcTN <- simComponents(eusilcS, eusilcTN, w = "rb050",
+     total = "netIncome", components = components,
+     conditional = c("netIncomeCat", "pl030"), seed = seedTN)
```

Finally, diagnostic box-and-whisker plots of the income components are produced with the function `spBwplot()`. Since the box widths correspond to the ratio of non-zero observations to the total number of observed values and most of the components contain large proportions of zeros, a minimum box width is specified using the argument `minRatio`. Figure 3.4 contains the resulting plots.

```
R> listP <- list(MP = eusilcMP, TR = eusilcTR, TN = eusilcTN)
R> spBwplot(components, "rb050", dataS = eusilcS, dataP = listP,
+     pch = "|", minRatio = 0.2, layout = c(2, 4))
```

3.4 Conclusions

In this vignette, the use of `simPopulation` for simulating population data has been demonstrated in an application to the EU-SILC example data from the package. Both the simulation of synthetic population data and the generation of diagnostic plots have been illustrated in a similar analysis as in ALFONS et al. (2010).

The code examples show that the functions are easy to use and that the arguments have sensible default values. Nevertheless, the behavior of the functions is highly customizable. In particular the functions for the diagnostic plots benefit from the implementations based on the packages `vcd` and `lattice`.

Bibliography

Alfons, A. and **Kraft, S.** (**2010**): `simPopulation`: Simulation of Synthetic Populations for Surveys based on Sample Data. R package version 0.2.1.
URL http://CRAN.R-project.org/package=simPopulation

Alfons, A., **Kraft, S.**, **Templ, M.** and **Filzmoser, P.** (**2010**): *Simulation of Synthetic Population Data for Household Surveys with Application to EU-SILC*. Research Report CS-2010-1, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2010-1complete.pdf

Eurostat (**2004**): *Description of Target Variables: Cross-sectional and Longitudinal*. EU-SILC 065/04, Eurostat, Luxembourg.

Meyer, D., **Zeileis, A.** and **Hornik, K.** (**2006**): *The `strucplot` Framework: Visualizing Multi-way Contingency Tables with `vcd`*. Journal of Statistical Software, 17 (3), pp. 1–48.

Meyer, D., **Zeileis, A.** and **Hornik, K.** (**2010**): `vcd`: Visualizing Categorical Data. R package version 1.2-9.
URL http://CRAN.R-project.org/package=vcd

R Development Core Team (**2010**): R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Sarkar, D. (**2008**): Lattice: Multivariate Data Visualization with R. New York: Springer, ISBN 978-0-387-75968-5.

Sarkar, D. (**2010**): `lattice`: Lattice Graphics. R package version 0.19-13.
URL http://CRAN.R-project.org/package=lattice

Chapter 4

An application of VIM to EU-SILC data

Abstract: Package VIM allows to explore and to analyze the structure of missing values in data, as well as to produce high-quality graphics for publications. This paper illustrates an application of VIM to a highly complex data set – the European Statistics on Income and Living Conditions (EU-SILC).

Keywords: Missing Values Exploration, Visualization, R

4.1 The graphical user interface of VIM

The graphical user interface (GUI) has been developed using the R package tcltk (R Development Core Team, 2009) and allows easy handling of the functions included in package VIM. Figure 4.1 shows the GUI, which pops up automatically after loading the package.

```
> library(VIM)
```

If the GUI has been closed, it can be reopened with the following command. All selections and settings from the last session are thereby recovered.

```
> vmGUImenu()
```

For visualization, the most important menus are the *Data*, the *Visualization* and the *Options* menus.

4.1.1 Handling data

The *Data* menu allows to select a data frame from the R workspace (see Figure 4.2). In addition, a data set in .RData format can be imported from the file system into the R workspace, which is then loaded into the GUI directly.

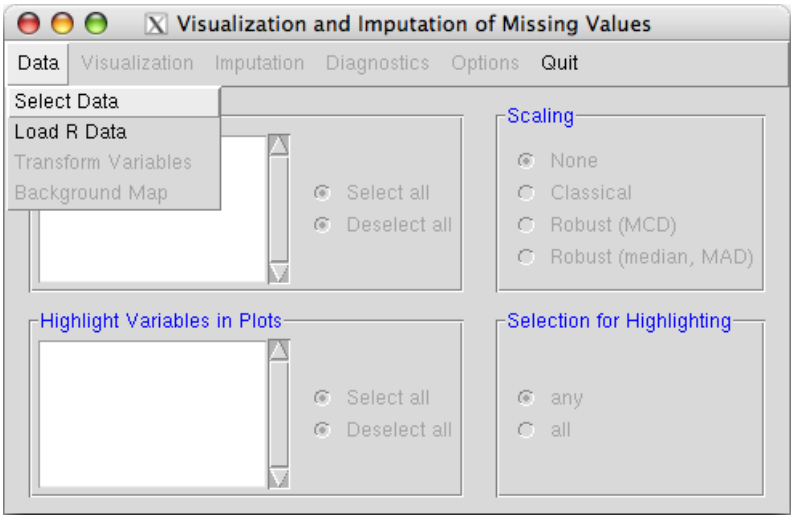

Figure 4.1: The VIM GUI and the *Data* menu.

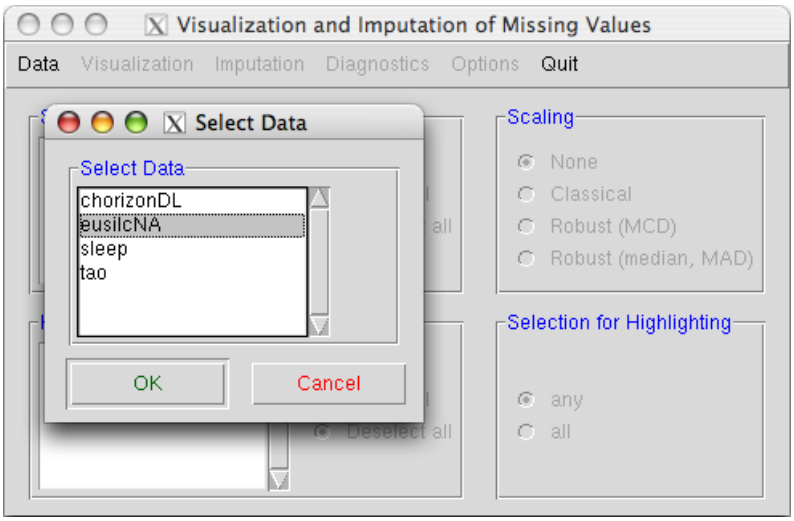

Figure 4.2: The dialog for data selection.

Transformations of variables are available via `Data` → `Transform Variables`. The transformed variables are thereby appended to the data set in use. Commonly used transformations in official statistics are available, e.g., the Box-Cox transformation (Box and Cox, 1964) and the log-transformation as an important special case of the Box-Cox transformation. In addition, several other transformations that are frequently used for compositional data (Aitchison, 1986) are implemented. Background maps and coordinates for spatial data can be selected in the *Data* menu as well.

Functionality to scale variables, on the other hand, is offered in the upper right frame of the GUI. Note that scaling is performed on-the-fly, i.e., the scaled variables are simply passed to the underlying plot functions, they are not permanently stored.

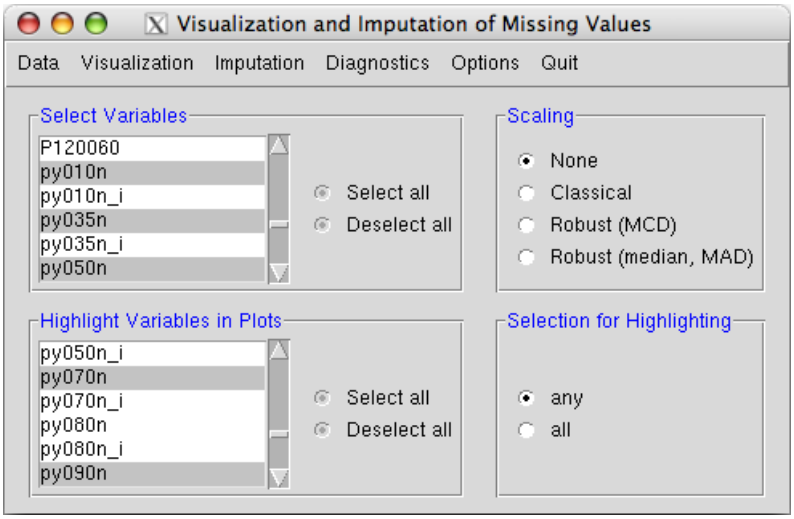

Figure 4.3: Variable selection with the `VIM` GUI.

4.1.2 Selecting variables

After a data set has been chosen, variables can be selected in the main dialog (see Figure 4.3). An important feature is that the variables will be used in the same order as they were selected, which is especially useful for parallel coordinate plots.

Variables for highlighting are distinguished from the plot variables and can be selected separately (see the lower left frame in Figure 4.3). If more than one variable chosen for highlighting, it is possible to select whether observations with missing values in any or in all of these variables should be highlighted (see in the lower right frame in Figure 4.3).

4.1.3 Selecting plots

A plot method can be selected from the *Visualization* menu. Note that plots that are not applicable with the selected variables are disabled, e.g., if only one plot variable is selected, multivariate plots are not available.

4.2 An application to EU-SILC data

In this section, some of the visualization tools are illustrated on the public use sample of the Austrian EU-SILC data from 2004 (Statistics Austria, 2007), which can be obtained from Statistics Austria (see Table 4.1 for an explanation of the variables used here). This well-known and complex data set is mainly used for measuring risk-of-poverty and social cohesion in Europe, and for monitoring the Lisbon 2010 strategy of the European Union. The raw data set contains a high amount of missing values, which are imputed with model-based and donor-based imputation methods before public release (Statistics Austria, 2006). Since a high amount of missing values are not MCAR, the variables to

Table 4.1: Explanation of the used variables from the EU-SILC data set.

name	meaning
age	Age
R007000	Occupation
P033000	Years of employment
py010n	Employee cash or near cash income
py035n	Contributions to individual private pension plans
py050n	Cash benefits or losses from self-employment
py070n	Values of goods produced by own-consumption
py080n	Pension from individual private plans
py090n	Unemployment benefits
py100n	Old-age benefits
py110n	Survivors' benefits
py120n	Sickness benefits
py130n	Disability benefits
py140n	Education-related allowances

be included for imputation need to be selected carefully. This problem can be solved with our proposed visualization tools.

```
> incvars <- c(paste("py", c("010", "035", "050", "070", "080",
+     "090", "100", "110", "120", "130", "140"), "n", sep=""))
> eusilcNA[, incvars] <- log10(eusilcNA[, incvars] + 1)
```

First of all, it may be of interest how many missing values are contained in each variable. Even more interesting, missing values may frequently occur in certain combinations of variables. This can easily investigated by selecting variables of interest (see Figure 4.3) and by clicking on `Visualization` → `Aggregate Missings`. If one prefers the command line language of R, the the plot in Figure 4.4 can be created by invoking:

```
> aggr(eusilcNA[, incvars], numbers=TRUE, prop = c(TRUE, FALSE))
```

Here `eusilcNA` denotes the data frame in use (see also Figure 4.2). The barplot on the left hand side shows the proportion of missing values in each of the selected variables. On the right hand side, all existing combinations of missing and non-missing values in the observations are visualized. A red rectangle indicates missingness in the corresponding variable, a blue rectangle represents available data. In addition, the frequencies of the different combinations are represented by a small bar plot and by numbers. Variables may be sorted by the number of missing values and combinations by the frequency of occurrence to give more power to finding the structure of missing values. For example, the top row in Figure 4.4 (right) represents the combination with missing values in variables *py010n* (employee cash or near cash income), *py035n* (contributions to individual private pension plans) and *py090n* (unemployment benefits), and observed values in the remaining variables, which appears only once in the data.

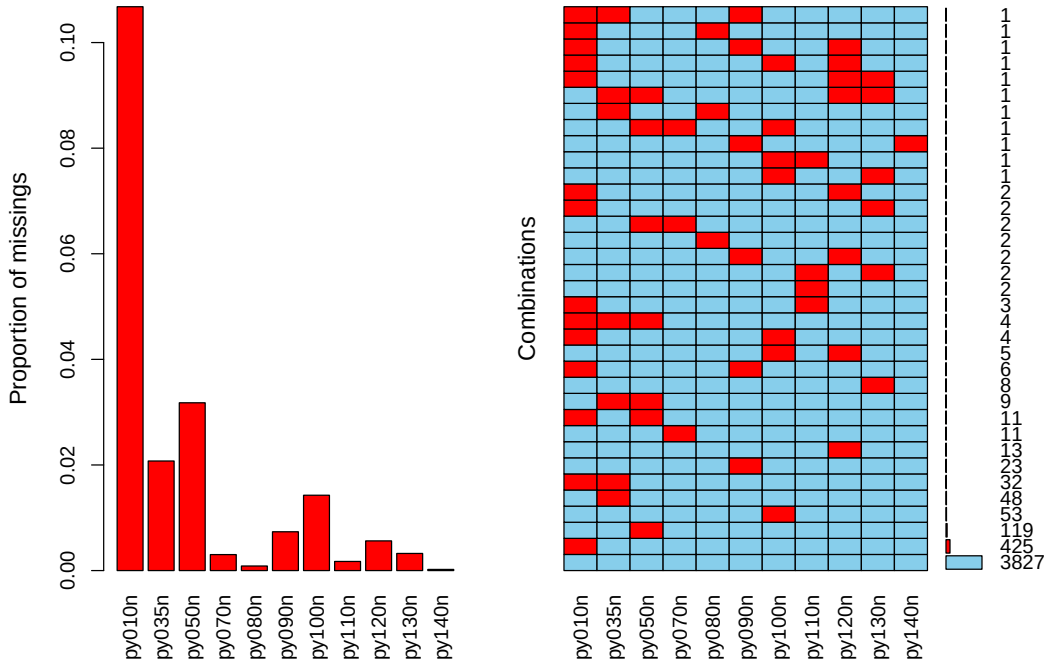

Figure 4.4: Aggregation plot of the income components in the public use sample of the Austrian EU-SILC data from 2004. *Left*: barplot of the proportions of missing values in each of the income components. *Right*: all existing combinations of missing (red) and non-missing (blue) values in the observations. The frequencies of the combinations are visualized by small horizontal bars.

The plot reveals an exceptionally high number of missing values in variable *py010n*. The combination with variable *py035n* still contains 32 missing values. Note that it is possible to display proportions of missing values and combinations rather than absolute numbers.

4.2.1 Univariate plots

When only one variable is selected, only plots emphasized in Figure 4.5 can be applied. Standard univariate plots, such as barplots and spine plots for categorical variables and histograms, spinograms and different types of boxplots for continuous variables, have been adapted to display information about missing values.

For example, it may be of interest to display the distribution of years of employment, with missing values in *py010n* (employee cash or near cash income) highlighted. A spinogram (Hofmann and Theus, 2005) can easily be generated by clicking `Visualization` → `Spinogram with Missings`. Alternatively, the output shown in Figure 4.6 can be produced with the following command:

```
> spineMiss(eusilcNA[, c("P033000", "py010n")])
```

Figure 4.6 indicates that the probability of missingness in *py010n* depends on the years of employment.

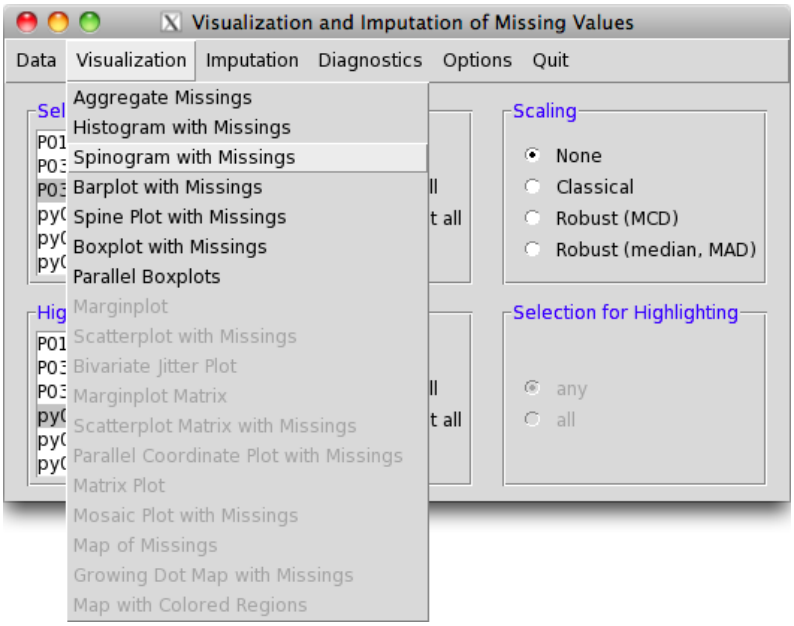

Figure 4.5: Univariate Plots supported by the VIM GUI.

4.2.2 Bivariate plots

For bivariate data, different kinds of scatterplots are implemented. Figure 4.7 lists the plots applicable when two variables are selected. Multivariate plots are also highlighted because they can be used in the bivariate case, too.

Figure 4.8 shows a scatterplot with information about the univariate distributions and missingness of the variables in the plot margins (`Visualization` → `Marginplot`). The boxplots in red indicate observations with missing values in the other variable. It is clearly visible that the amount of missingness in *py010n* (employee cash or near cash income) is less for older people. Note that semi-transparent colors are used to prevent overplotting. The figure can also be produced with the command line interface of R, using the following command:

```
> marginplot(eusilcNA[, c("age", "py010n")], alpha = 0.6)
```

4.2.3 Multivariate plots

Parallel coordinate plots (Wegman, 1990) are very powerful for displaying multivariate relationships in data. A natural way of displaying information about missing data is to highlight observations according to missingness in a certain variable or a combination of variables. However, plotting variables with missing values results in disconnected lines, making it impossible to trace the respective observations across the graph. As a remedy, missing values may be represented by a point above the corresponding coordinate axis, which is separated from the main plot by a small gap and a horizontal line (see Figure 4.9). Connected lines can then be drawn for all observations.

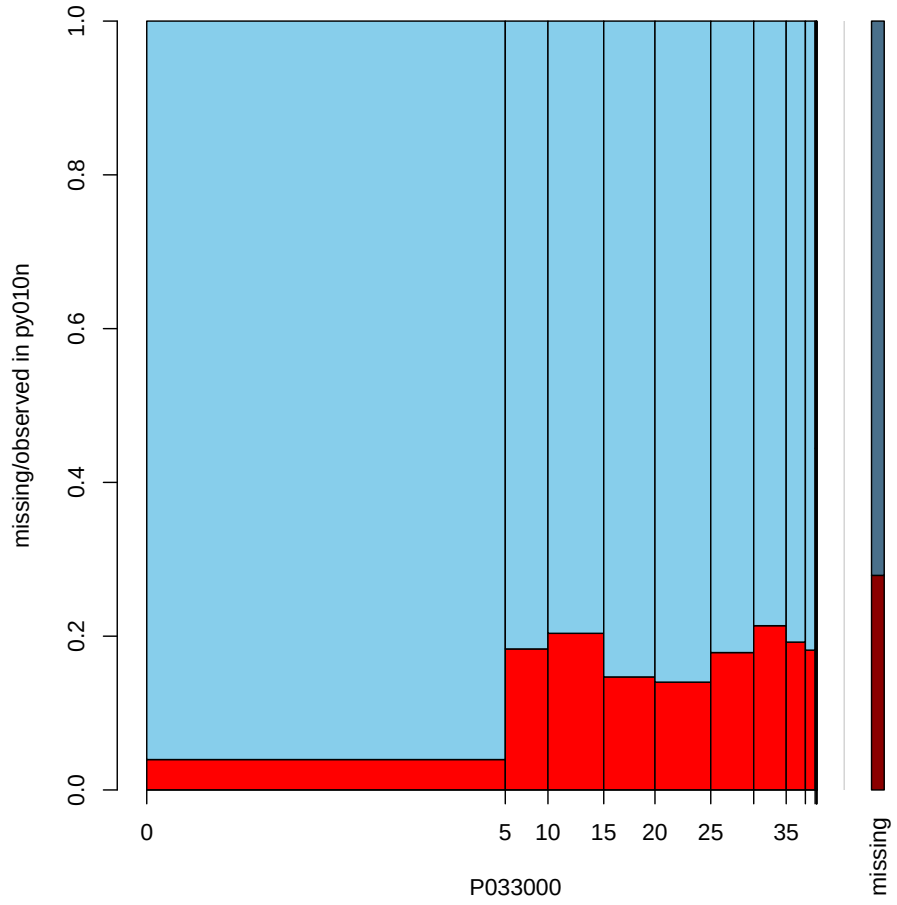

Figure 4.6: Spinogram of *P033000* (years of employment) with color coding for missing (red) and available (blue) data in *py010n* (employee cash or near cash income).

Such parallel coordinate plots can be genereated by clicking `Visualization` → `Parallel Coordinate Plot with Missings` in the GUI or by using the function `parcoordMiss()` on the command line. The example in Figure 4.9 can be produced with:

```
> parcoordMiss(eusilcNA[, c("age", "P033000", "py010n", "py035n",
+     "py050n")], plotvars = 1:4, highlight = 5, alpha = 0.2)
```

A data frame containing all variables of interest needs to supplied as the first argument, the variables to be plotted are given by argument `plotvars`, and the variables to be used for highlighting are specified by argument `highlight`.

Due to the large number of lines, a very low alpha value (i.e., very high transparancy) is used in Figure 4.9 to prevent overplotting. Missing values in *py050n* occur mainly for middle-aged people. Moreover, observations with missing values in *py050n* behave in an entire different way for the variables *py010n* (employee cash or near cash income) and *py035n* (contributions to individual private pension plans) than the main part of the data.

The *matrix plot* is an even more powerful multivariate plot. It visualizes all cells of the data matrix by (small) rectangles. In the example in Figure 4.10, red rectangles are

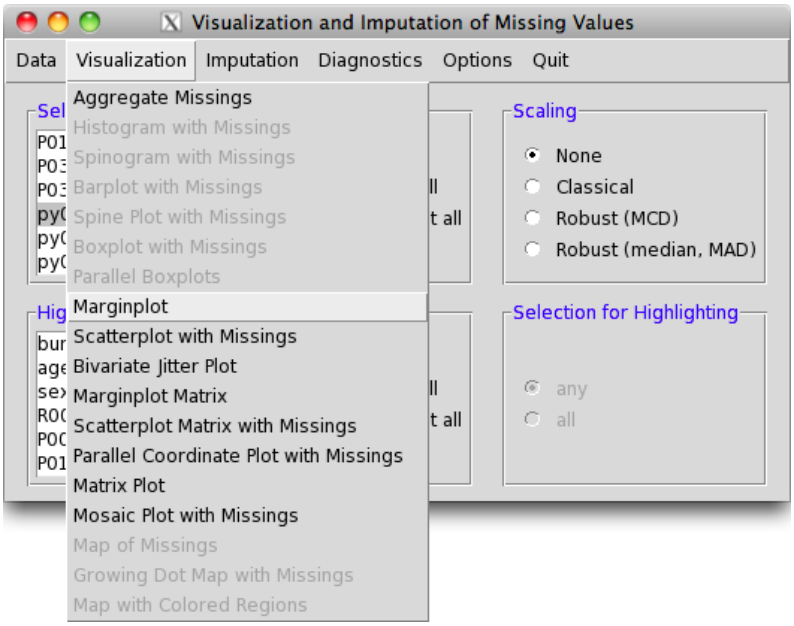

Figure 4.7: Bivariate plots (as well as multivariate plots) available in the VIM GUI.

drawn for missing values, and a greyscale is used for the available data. To determine the grey level, the variables are first scaled to the interval $[0, 1]$. Small values are assigned a light grey and high values a dark grey (0 corresponds to white, 1 to black). In addition, the observations can be sorted by the magnitude of a selected variable, which can also be done interactively by clicking in the corresponding column of the plot. Using the GUI, a matrix plot can be produced by clicking `Visualization` → `Matrix Plot`. The example in Figure 4.10 can also be created on the command line by invoking the following command:

```
> matrixplot(eusilcNA[, c("age", "R007000", incvars)],
+     sortby = "R007000")
```

Figure 4.10 shows a matrix plot of *age*, *R007000* (occupation) and the transformed income components, sorted by variable *R007000* (occupation). It is clearly visible that missing values in most income components depend on the occupation of the corresponding person. Thus the missing data mechanism was found to be MAR for these variables, which should be considered when applying imputation methods.

4.2.4 Other plots

Various other plots are availabe in the package and can also be created with the GUI (see Figures 4.5 and 4.7). For spatial data, mapping is supported if a background map is provided by the user, e.g., as a shape file, data frame or list of coordinates.

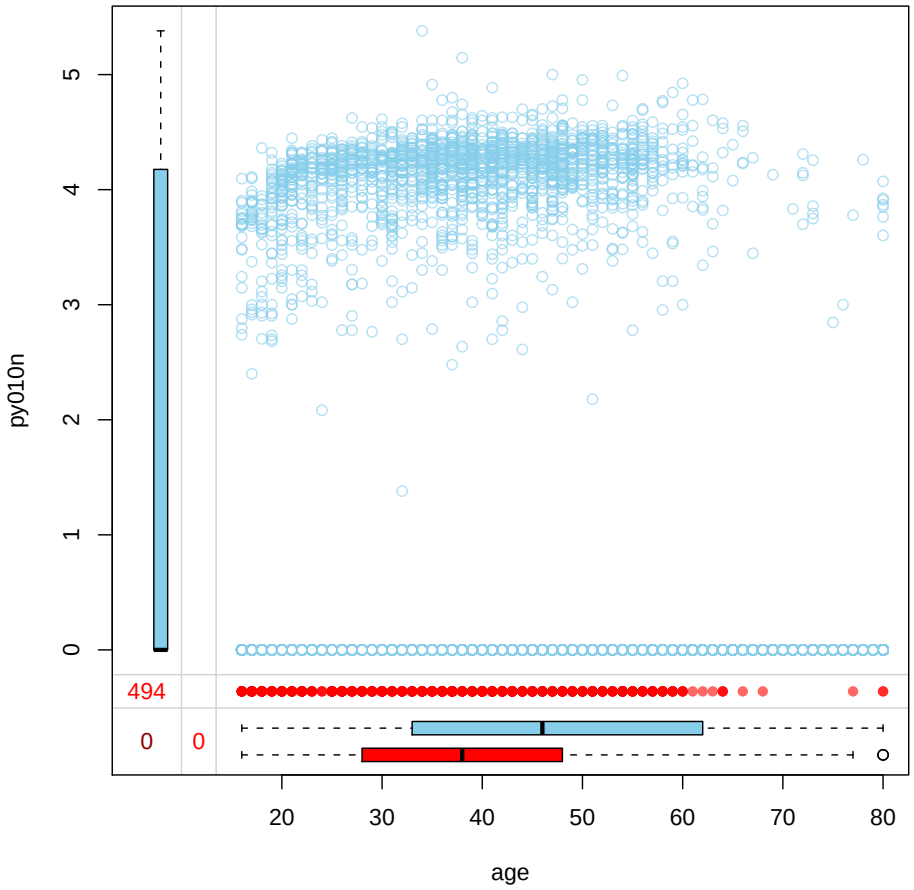

Figure 4.8: Scatterplot of *age* and transformed *py010n* (employee cash or near cash income) with information about missing values in the plot margins.

4.3 Fine tuning

In the *Preferences* dialog from the *Options* menu (click `Options` → `Preferences`), which is displayed in Figure 4.11, the colors and alpha channel to be used in the plots can be set. In addition, it contains an option to embed multivariate plots in `Tcl/Tk` windows. This is useful if the number of observations and/or variables is large, because scrollbars allow to move from one part of the plot to another.

4.4 Interactive features

Many interactive features are implemented in the plot functions in order to allow easy modification of the plots.

When variables are selected for highlighting in univariate plots such as histograms, barplots, spine plots or spinograms, it is possible to switch between the variables. Clicking in the right plot margin of a histogram, for example, corresponds with creating a histogram

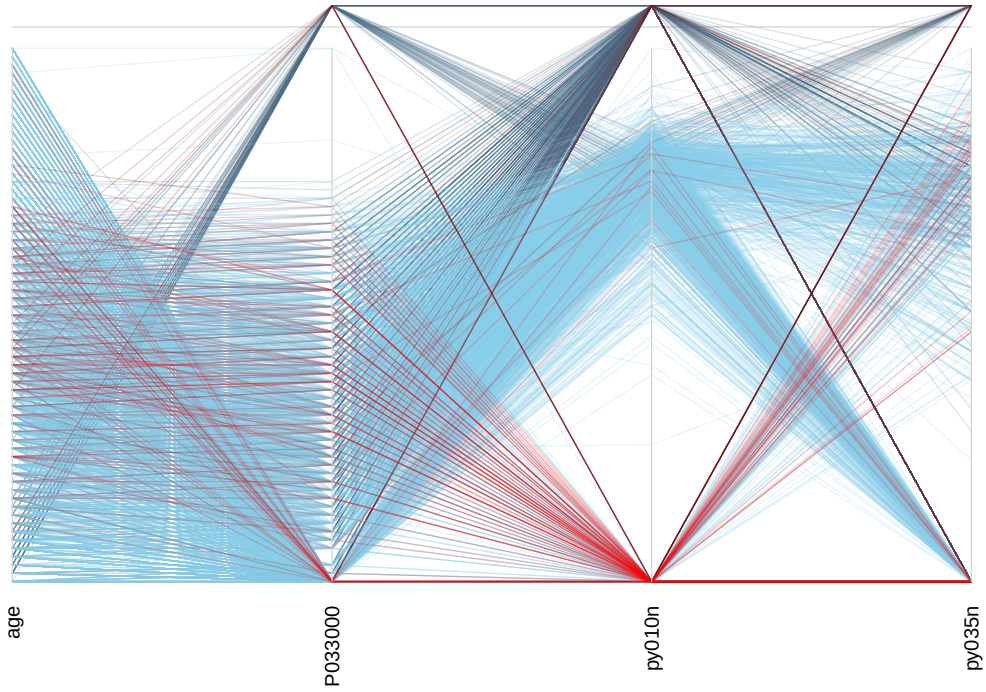

Figure 4.9: Parallel coordinate plot of *age*, *P033000* (years of employment), transformed *py010n* (employee cash or near cash income) and transformed *py035n* (contributions to individual private pension plans), with color coding for missing (red) and available (blue) data in variable *py050n* (cash benefits or losses from self-employment).

(or barplot) for the next variable, and clicking in the left margin switches to the previous variable. This interactive feature is particularly usedful for parallel boxplots, as it allows to view all possible $p(p-1)$ combinations with $p-1$ clicks, where p denotes the number of variables.

For multivariate plots (scatterplot matrix and parallel coordinate plot), variables for highlighting can be selected and deselected interactively, by clicking in a diagonal panel of the scatterplot matrix or on a coordinate axis in the parallel coordinate plot. Information about the current selection is printed on the R-console.

The matrixplot is particulary powerful if the observations are sorted by a specific variable (see Figure 4.10). This can be done by clicking on the corresponding column.

4.5 Summary

We showed that the visualization of missing values is extremely simple with package `VIM`, either by using the GUI or by typing code on the R command line. With the visualization techniques in `VIM`, it is possible to gain insight into the data and to understand the structure of missing values. The latter is absolutely necessary when dealing with missing values, e.g., before imputation is performed.

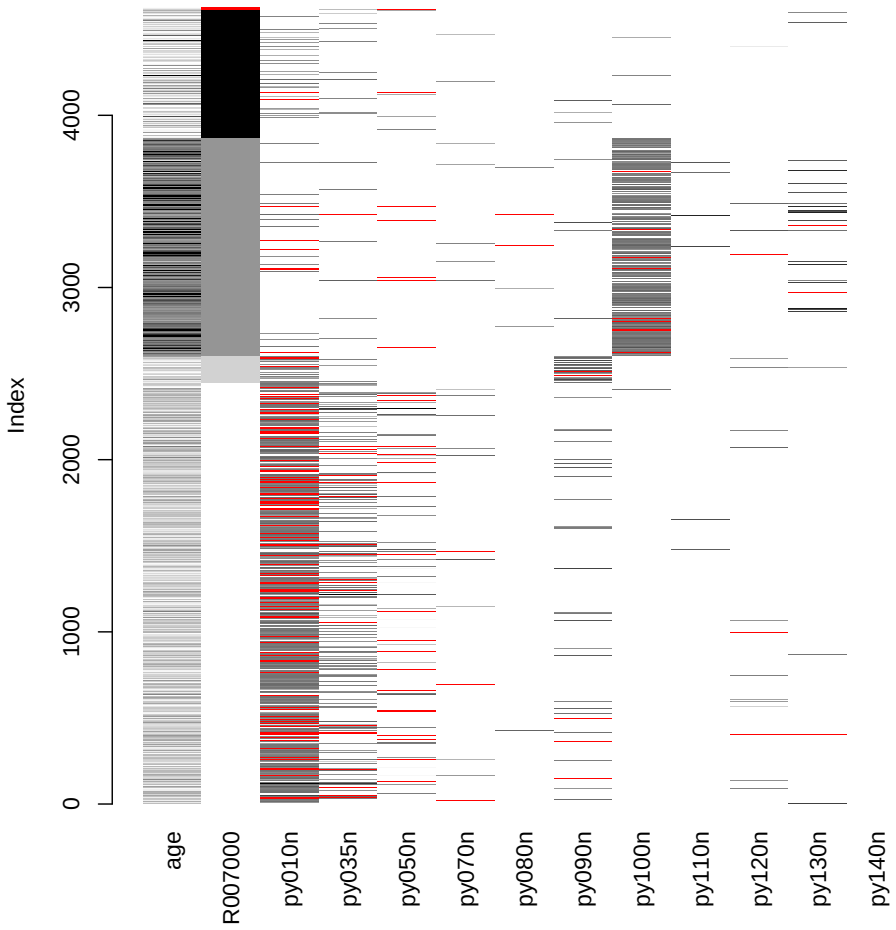

Figure 4.10: Matrixplot of *age*, *R007000* (occupation) and the transformed income components, sorted by variable *R007000* (occupation).

Bibliography

Aitchison, J. (**1986**): The Statistical Analysis of Compositional Data. Wiley, New York.

Box, G. and **Cox, D.** (**1964**): *An Analysis of Transformations*. Journal of the Royal Statistical Society, Series B, 26, pp. 211–252.

Hofmann, H. and **Theus, M.** (**2005**): *Interactive graphics for visualizing conditional distributions*, unpublished manuscript.

R Development Core Team (**2009**): R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Statistics Austria (**2006**): *Einkommen, Armut und Lebensbedingungen 2004, Ergebnisse aus EU-SILC 2004*. In German. ISBN 3-902479-59-0.

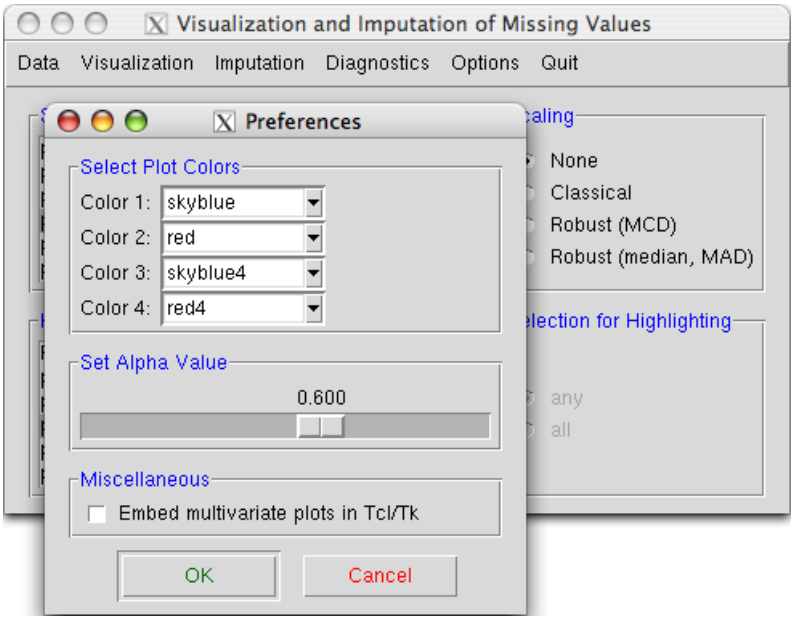

Figure 4.11: The *Preferences* dialog of the `VIM` GUI.

Statistics Austria (**2007**): *EU-SILC 2004. Erläuterungen: Mikrodaten-Subsample für externe Nutzer*. In German.

Wegman, E. (**1990**): *Hyperdimensional data analysis using parallel coordinates*. Journal of the American Statistical Association, 85 (411), pp. 664–675.

Chapter 5

Standard Methods for Social Exclusion Indicators in package `laeken`

Abstract This vignette demonstrates the use of the R package `laeken` for standard point estimation of indicators on social exclusion and poverty according to the definitions by Eurostat. The package contains synthetically generated data for the European Union Statistics on Income and Living Conditions (EU-SILC), which is used in the code examples throughout the paper. Furthermore, the basic object-oriented design of the package is discussed. Even though the paper is focused on showing the functionality of package `laeken`, it also provides a brief mathematical description of the implemented indicators.

Keywords: Social inclusion indicators, software, R

5.1 Introduction

The *European Union Statistics on Income and Living Conditions* (EU-SILC) is a panel survey conducted in EU member states and other European countries, and serves as basis for measuring risk-of-poverty and social cohesion in Europe. A short overview of the 11 most important indicators on social exclusion and poverty according to Eurostat (2004) is given in the following.

Primary indicators

1. At-risk-of-poverty rate (after social transfers)
2. At-risk-of-poverty rate by age and gender
3. At-risk-of-poverty rate by most frequent activity status and gender
4. At-risk-of-poverty rate by household type
5. At-risk-of-poverty rate by accommodation tenure status
6. At-risk-of-poverty rate by work intensity of the household

7. At-risk-of-poverty threshold (illustrative values)
8. Inequality of income distribution: S80/S20 income quintile share ratio
9. At-persistent-risk-of-poverty rate by age and gender (60% median)
10. Relative median at-risk-of-poverty gap, by age and gender

Secondary indicators

11. Dispersion around the at-risk-of-poverty threshold
12. At-risk-of-poverty rate anchored at a moment in time
13. At-risk-of-poverty rate before social transfers by age and gender
14. Inequality of income distribution: Gini coefficient
15. At-persistent-risk-of-poverty rate, by age and gender (50% median)

Other indicators

16. Mean equivalized disposable income
17. The gender pay gap

Note that especially the Gini coefficient is very well studied due to its importance in many fields of research.

The add-on package `laeken` (Alfons et al., 2011a) aims is to bring functionality for the estimation of indicators on social exclusion and poverty to the statistical environment R (R Development Core Team, 2011). In the examples in this vignette, standard estimates for the most important indicators are computed according to the Eurostat definitions (Eurostat, 2004, 2009). More sophisticated methods that are less influenced by outliers are described in vignette `laeken-pareto` (Alfons et al., 2011c), while the basic framework for variance estimation is discussed in vignette `laeken-variance` (Templ and Alfons, 2011). Those documents can be viewed from within R with the following commands:

```
R> vignette("laeken-pareto")
R> vignette("laeken-variance")
```

The example data set of package `laeken`, which is called `eusilc` and consists of 14 827 observations from 6 000 households, is used throughout the paper. It was synthetically generated from Austrian EU-SILC survey data from 2006 using the data simulation methodology proposed by Alfons et al. (2011b) and implemented in the R package `simPopulation` (Alfons and Kraft, 2010). The first three observations of the synthetic data set `eusilc` are printed below.

```
R> library("laeken")
R> data("eusilc")
R> head(eusilc, 3)
```

```
  db030 hsize db040 rb030 age  rb090 pl030 pb220a   py010n py050n py090n py100n
1     1     3 Tyrol   101  34 female     2     AT  9756.25      0      0      0
2     1     3 Tyrol   102  39   male     1  Other 12471.60      0      0      0
3     1     3 Tyrol   103   2   male  <NA>   <NA>       NA     NA     NA     NA
  py110n py120n py130n py140n hy040n  hy050n hy070n hy080n hy090n hy110n hy130n
1      0      0      0      0 4273.9 2428.11      0      0  33.39      0      0
2      0      0      0      0 4273.9 2428.11      0      0  33.39      0      0
3     NA     NA     NA     NA 4273.9 2428.11      0      0  33.39      0      0
  hy145n eqSS eqIncome    db090    rb050
1      0  1.8 16090.69 504.5696 504.5696
2      0  1.8 16090.69 504.5696 504.5696
3      0  1.8 16090.69 504.5696 504.5696
```

Only a few of the large number of variables in the original survey are included in the example data set. The variable names are rather cryptic codes, but these are the standardized names used by the statistical agencies. Furthermore, the variables `hsize` (household size), `age`, `eqSS` (equivalized household size) and `eqIncome` (equivalized disposable income) are not included in the standardized format of EU-SILC data, but have been derived from other variables for convenience. Moreover, some very sparse income components were not included in the the generation of this synthetic data set. Thus the equivalized household income is computed from the available income components.

For the remainder of the paper, the variable `eqIncome` (equivalized disposable income) is of main interest. Other variables are in some cases used to break down the data in order to evaluate the indicators on the resulting subsets.

It is important to note that EU-SILC data are in practice conducted through complex sampling designs with different inclusion probabilities for the observations in the population, which results in different weights for the observations in the sample. Furthermore, calibration is typically performed for non-response adjustment of these initial design weights. Therefore, the sample weights have to be considered for all estimates, otherwise biased results are obtained.

The rest of the paper is organized as follows. Section 5.2 briefly illustrates the basic object-oriented design of the package. The calculation of the equivalized household size and the equivalized disposable income is then described in Section 5.3. Afterwards, Section 5.4 introduces the Eurostat definitions of the weighted median and weighted quantiles, which are required for the estimation of some of the indicators. In Section 5.5, a mathematical description of the most important indicators on social exclusion and poverty is given and their estimation with package `laeken` is demonstrated. Section 5.6 discusses a useful subsetting method, and Section 7.5 concludes.

5.2 Basic design of the package

The implementation of the package follows an object-oriented design using S3 classes (Chambers and Hastie, 1992). Its aim is to provide functionality for point and variance estimation of Laeken indicators with a single command, even for different years and domains. Currently, the following indicators are available in the R package `laeken`:

- *At-risk-of-poverty rate*: function `arpr()`
- *Quintile share ratio*: function `qsr()`
- *Relative median at-risk-of-poverty gap*: function `rmpg()`
- *Dispersion around the at-risk-of-poverty threshold*: also function `arpr()`
- *Gini coefficient*: function `gini()`

Note that the implementation strictly follows the Eurostat definitions (Eurostat, 2004, 2009).

5.2.1 Class structure

In this section, the class structure of package `laeken` is briefly discussed. Section 5.2.1 describes the basic class `'indicator'`, while the different subclasses for the specific indicators are listed in Section 5.2.1.

Class `'indicator'`

The basic class `'indicator'` acts as the superclass for all classes in the package corresponding to specific indicators. It consists of the following components:

`value`: A numeric vector containing the point estimate(s).

`valueByStratum`: A `data.frame` containing the point estimates by stratum.

`varMethod`: A character string specifying the type of variance estimation used.

`var`: A numeric vector containing the variance estimate(s).

`varByStratum`: A `data.frame` containing the variance estimates by stratum.

`ci`: A numeric vector or matrix containing the confidence interval(s).

`ciByStratum`: A `data.frame` containing the confidence intervals by stratum.

`alpha`: The confidence level is given by 1−`alpha`.

`years`: A numeric vector containing the different years of the survey.

`strata`: A character vector containing the different strata of the breakdown.

These list components are inherited by each indicator in the package. One of the most important features of `laeken` is that indicators can be evaluated for different years and domains. The latter of which can be regions (e.g., NUTS2), but also any other breakdown given by a categorical variable (see the examples in Section 5.5).

In any case, the advantage of the object-oriented implementation is the possibility of sharing code among the indicators. To give an example, the following methods for the basic class `'indicator'` are implemented in the package:

```
R> methods(class = "indicator")
```

```
[1] bootVar.indicator* print.indicator*   subset.indicator*

   Non-visible functions are asterisked
```

The `print()` and `subset()` methods are called by their respective generic functions if an object inheriting from class `'indicator'` is supplied. While the `print()` method defines the output of objects inheriting from class `'indicator'` shown on the R console, the `subset()` method allows to extract subsets of an object inheriting from class `'indicator'` and is discussed in detail in Section 5.6. Furthermore, the function `is.indicator()` is available to test whether an object is of class `'indicator'`.

Additional classes

For the specific indicators on social exclusion and poverty, the following classes are implemented in package `laeken`:

- Class `'arpr'` with the following additional components:

  `p`: The percentage of the weighted median used for the at-risk-of-poverty threshold.

  `threshold`: The at-risk-of-poverty threshold(s).

- Class `'qsr'` with no additional components.

- Class `'rmpg'` with the following additional components:

  `threshold`: The at-risk-of-poverty threshold(s).

- Class `'gini'` with no additional components.

All these classes are subclasses of the basic class `'indicator'` and therefore inherit all its components and methods. In addition, functions to test whether an object is a member of one of these subclasses are implemented. Similarly to `is.indicator()`, these are called `is.foo()`, where `foo` is the name of the respective class (e.g., `is.arpr()`).

5.3 Calculation of the equivalized disposable income

For each person, the equivalized disposable income is defined as the total household disposable income divided by the equivalized household size. It follows that each person in the same household receives the same equivalized disposable income.

The total disposable income of a household is calculated by adding together the personal income received by all of the household members plus the income received at the household level. The equivalized household size is defined according to the modified OECD scale, which gives a weight of 1.0 to the first adult, 0.5 to other household members aged 14 or over, and 0.3 to household members aged less than 14 (Eurostat, 2004, 2009).

In practice, the equivalized disposable income needs to be computed from the income components included in EU-SILC for the estimation of the indicators on social exclusion and poverty. Therefore, this section outlines how to perform this step with package

laeken, even though the variable eqIncome containing the equivalized disposable income is already available in the example data set eusilc. Note that not all variables that are required for an exact computation of the equivalized income are included in the synthetic example data. However, the functions of the package can be applied in exactly the same manner to real EU-SILC data.

First, the equivalized household size according to the modified OECD scale needs to be computed. This can be done with the function eqSS(), which requires the household ID and the age of the individuals as arguments. In the example data, household ID and age are stored in the variables db030 and age, respectively. It should be noted that the variable age is not in the standardized format of EU-SILC data and needs to be calculated from the data beforehand. Nevertheless, these computations are very simple and are therefore not shown here (for details, see Eurostat, 2009). The following two lines of code calculate the equivalized household size, add it to the data set, and print the first eight observations of the variables involved.

```
R> eusilc$eqSS <- eqSS("db030", "age", data = eusilc)
R> head(eusilc[, c("db030", "age", "eqSS")], 8)

  db030 age eqSS
1     1  34  1.8
2     1  39  1.8
3     1   2  1.8
4     2  38  2.1
5     2  43  2.1
6     2  11  2.1
7     2   9  2.1
8     3  26  1.0
```

Then the equivalized disposable income can be computed with the function eqInc(). It requires the following information to be supplied: the household ID, the household income components to be added and subtracted, respectively, the personal income components to be added and subtracted, respectively, as well as the equivalized household size. With the following commands, the equivalized disposable income is calculated and added to the data set, after which the first eight observations of the important variables in this context are printed.

```
R> hplus <- c("hy040n", "hy050n", "hy070n", "hy080n", "hy090n",
+     "hy110n")
R> hminus <- c("hy130n", "hy145n")
R> pplus <- c("py010n", "py050n", "py090n", "py100n", "py110n",
+     "py120n", "py130n", "py140n")
R> eusilc$eqIncome <- eqInc("db030", hplus, hminus, pplus, character(),
+     "eqSS", data = eusilc)
R> head(eusilc[, c("db030", "eqSS", "eqIncome")], 8)

  db030 eqSS eqIncome
1     1  1.8 16090.69
2     1  1.8 16090.69
```

```
3      1  1.8 16090.69
4      2  2.1 27076.24
5      2  2.1 27076.24
6      2  2.1 27076.24
7      2  2.1 27076.24
8      3  1.0 19659.53
```

Note that the net income is considered in this example, therefore no personal income component needs to be subtracted (see Eurostat, 2004, 2009). This is reflected in the call to `eqInc()` by the use of an empty character vector `character()` for the corresponding argument.

5.4 Weighted median and quantile estimation

Some of the indicators on social exclusion and poverty require the estimation of the median income or other quantiles of the income distribution. Hence functions that strictly follow the definitions according to Eurostat (2004, 2009) are implemented in package `laeken`. They are used internally for the estimation of the respective indicators, but can also be called by the user directly.

In the analysis of income distributions, the median income is typically of higher interest than the arithmetic mean. This is because income distributions commonly are strongly right-skewed with a heavy tail of *representative outliers* (correctly measured units that are not unique to the population) and *nonrepresentative outliers* (either measurement errors or correct observations that can be considered unique in the population). Therefore, the center of the distribution is more reliably estimated by a weighted median than by a weighted mean, as the latter is highly influenced by extreme values.

In mathematical terms, quantiles are defined as $q_p := F^{-1}(p)$, where F is the distribution function on the population level and $0 \leq p \leq 1$. The median as an important special case is given by $p = 0.5$. For the following definitions, let n be the number of observations in the sample, let $\boldsymbol{x} := (x_1, \ldots, x_n)'$ denote the equivalized disposable income with $x_1 \leq \ldots \leq x_n$, and let $\boldsymbol{w} := (w_i, \ldots, w_n)'$ be the corresponding personal sample weights. Weighted quantiles for the estimation of the population values according to Eurostat (2004, 2009) are then given by

$$\hat{q}_p = \hat{q}_p(\boldsymbol{x}, \boldsymbol{w}) := \begin{cases} \frac{1}{2}(x_j + x_{j+1}), & \text{if } \sum_{i=1}^{j} w_i = p \sum_{i=1}^{n} w_i, \\ x_{j+1}, & \text{if } \sum_{i=1}^{j} w_i < p \sum_{i=1}^{n} w_i < \sum_{i=1}^{j+1} w_i. \end{cases} \tag{5.1}$$

This definition of weighted quantiles is available in `laeken` through the function `weightedQuantile()`. The following command computes the weighed 20% quantile, the weighted median, and the weighted 80% quantile. In the context of social exclusion indicators, these are of most importance.

```
R> weightedQuantile(eusilc$eqIncome, eusilc$rb050,
+     probs = c(0.2, 0.5, 0.8))
```

```
[1] 12212.60 18098.73 25997.65
```

For the important special case of the weighted median, the function `weightedMedian()` is available for convenience.

```
R> weightedMedian(eusilc$eqIncome, eusilc$rb050)
```

```
[1] 18098.73
```

In addition, the functions `incMedian()` and `incQuintile()` are more tailored towards application in the case of indicators on social exclusion and poverty and provide a similar interface as the functions for the indicators (see Section 5.5). In particular, they allow to supply an additional variable to be used as tie-breakers for sorting, and to compute the weighted median and income quintiles, respectively, for several years of the survey. With the following lines of code, the median income as well as the 1^{st} and 4^{th} income quintile (i.e., the weighted 20% and 80% quantiles) are estimated.

```
R> incMedian("eqIncome", weights = "rb050", data = eusilc)
```

```
[1] 18098.73
```

```
R> incQuintile("eqIncome", weights = "rb050", k = c(1, 4), data = eusilc)
```

```
       1        4
12212.60 25997.65
```

5.5 Indicators on social exclusion and poverty

In this section, the most important indicators on social exclusion and poverty are described in detail. Furthermore, the functionality of package `laeken` to estimate these indicators is demonstrated.

It should be noted that all functions for the implemented indicators provide a very similar interface. Most importantly, it is possible to compute estimates for several years of the survey and different subdomains with a single command. Furthermore, the functions allow to supply an additional variable to be used as tie-breakers for sorting. However, not all of the implemented functionality is shown in this vignette. For a complete description of the functions and their arguments, the reader is referred to the corresponding R help pages.

In addition, only point estimation of the indicators on social exclusion and poverty is illustrated here, statistical significance of these estimates is not discussed. The functionality for variance estimation of the indicators is described in the package vignette `laeken-variance` (Templ and Alfons, 2011).

For the following definitions of the estimators according to Eurostat (2004, 2009), let $\boldsymbol{x} := (x_1, \ldots, x_n)'$ be the equivalized disposable income with $x_1 \leq \ldots \leq x_n$ and let $\boldsymbol{w} := (w_i, \ldots, w_n)'$ be the corresponding personal sample weights, where n denotes the number of observations. Furthermore, define the following index sets for a certain threshold t:

$$I_{<t} := \{i \in \{1, \ldots, n\} : x_i < t\}, \tag{5.2}$$

$$I_{\leq t} := \{i \in \{1, \ldots, n\} : x_i \leq t\}, \tag{5.3}$$

$$I_{>t} := \{i \in \{1, \ldots, n\} : x_i > t\}. \tag{5.4}$$

5.5.1 At-risk-at-poverty rate

In order to define the *at-risk-of-poverty rate* (ARPR), the *at-risk-of-poverty threshold* (ARPT) needs to be introduced first, which is set at 60% of the national median equivalized disposable income. Then the at-risk-at-poverty rate is defined as the proportion of persons with an equivalized disposable income below the at-risk-at-poverty threshold (Eurostat, 2004, 2009). In a more mathematical notation, the at-risk-at-poverty rate is defined as

$$ARPR := P(x < 0.6 \cdot q_{0.5}) \cdot 100, \tag{5.5}$$

where $q_{0.5} := F^{-1}(0.5)$ denotes the population median (50% quantile) and F is the distribution function of the equivalized income on the population level.

For the estimation of the at-risk-at-poverty rate from a sample, the sample weights need to be taken into account. First, the at-risk-at-poverty threshold is estimated by

$$\widehat{ARPT} = 0.6 \cdot \hat{q}_{0.5}, \tag{5.6}$$

where $\hat{q}_{0.5}$ is the weighted median as defined in Equation (6.1). Then the at-risk-at-poverty rate can be estimated by

$$\widehat{ARPR} := \frac{\sum_{i \in I_{<\widehat{ARPT}}} w_i}{\sum_{i=1}^{n} w_i} \cdot 100, \tag{5.7}$$

where $I_{<\widehat{ARPT}}$ is an index set of persons with an equivalized disposable income below the estimated at-risk-of-poverty threshold as defined in Equation (5.2).

In package `laeken`, the functions `arpt()` and `arpr()` are implemented for the estimation of the at-risk-of-poverty threshold and the at-risk-of-poverty rate. Whenever sample weights are available in the data, they should be supplied as the `weights` argument. Even though `arpt()` is called internally by `arpr()`, it can also be called by the user directly.

```
R> arpt("eqIncome", weights = "rb050", data = eusilc)

[1] 10859.24

R> arpr("eqIncome", weights = "rb050", data = eusilc)

Value:
[1] 14.44422

Threshold:
[1] 10859.24
```

It is also possible to use these functions for the estimation of the indicator *dispersion around the at-risk-of-poverty threshold*, which is defined as the proportion of persons with an equivalized disposable income below 40%, 50% and 70% of the national weighted median equivalized disposable income. The proportion of the median equivalized income to be used can thereby be adjusted via the argument `p`.

```
R> arpr("eqIncome", weights = "rb050", p = 0.4, data = eusilc)

Value:
[1] 4.766885

Threshold:
[1] 7239.49

R> arpr("eqIncome", weights = "rb050", p = 0.5, data = eusilc)

Value:
[1] 7.988134

Threshold:
[1] 9049.363

R> arpr("eqIncome", weights = "rb050", p = 0.7, data = eusilc)

Value:
[1] 21.85638

Threshold:
[1] 12669.11
```

In order to compute estimates for different subdomains, a breakdown variable simply needs to be supplied as the `breakdown` argument. Note that in this case the same overall at-risk-of-poverty threshold is used for all subdomains (see EUROSTAT, 2004, 2009). The following command computes the overall estimate, as well as estimates for all NUTS2 regions.

```
R> arpr("eqIncome", weights = "rb050", breakdown = "db040", data = eusilc)

Value:
[1] 14.44422

Value by stratum:
        stratum    value
1    Burgenland 19.53984
2     Carinthia 13.08627
3 Lower Austria 13.84362
4      Salzburg 13.78734
5        Styria 14.37464
6         Tyrol 15.30819
7 Upper Austria 10.88977
8        Vienna 17.23468
9    Vorarlberg 16.53731

Threshold:
[1] 10859.24
```

However, any kind of breakdown can be supplied, e.g., the breakdowns defined by Eurostat (2004, 2009). With the following lines of code, a breakdown variable with all possible combinations of age categories and gender is defined and added to the data set, before it is used to compute estimates for the corresponding domains.

```
R> ageCat <- cut(eusilc$age, c(-1, 16, 25, 50, 65, Inf), right = FALSE)
R> eusilc$breakdown <- paste(ageCat, eusilc$rb090, sep = ":")
R> arpr("eqIncome", weights = "rb050", breakdown = "breakdown",
+     data = eusilc)

Value:
[1] 14.44422

Value by stratum:
           stratum     value
1   [-1,16):female 18.948125
2     [-1,16):male 17.973597
3   [16,25):female 16.703016
4     [16,25):male 16.156673
5   [25,50):female 15.220300
6     [25,50):male  9.638359
7   [50,65):female 12.941125
8     [50,65):male  8.221154
9  [65,Inf):female 21.252184
10   [65,Inf):male 12.046903

Threshold:
[1] 10859.24
```

Clearly, the results are even more heterogeneous than for the breakdown into NUTS2 regions.

5.5.2 Quintile share ratio

The income *quintile share ratio* (QSR) is defined as the ratio of the sum of the equivalized disposable income received by the 20% of the population with the highest equivalized disposable income to that received by the 20% of the population with the lowest equivalized disposable income (Eurostat, 2004, 2009).

For the estimation of the quintile share ratio from a sample, let $\hat{q}_{0.2}$ and $\hat{q}_{0.8}$ denote the weighted 20% and 80% quantiles, respectively, as defined in Equation (6.1). Using index sets $I_{\leq \hat{q}_{0.2}}$ and $I_{>\hat{q}_{0.8}}$ as defined in Equations (5.3) and (5.4), respectively, the quintile share ratio is estimated by

$$\widehat{QSR} := \frac{\sum_{i \in I_{>\hat{q}_{0.8}}} w_i x_i}{\sum_{i \in I_{\leq \hat{q}_{0.2}}} w_i x_i}. \tag{5.8}$$

With package `laeken`, the quintile share ratio can be estimated using the function `qsr()`. As for the at-risk-of-poverty rate, sample weights can be supplied via the `weights` argument.

```
R> qsr("eqIncome", weights = "rb050", data = eusilc)

Value:
[1] 3.971415
```

Computing estimates for different subdomains is again possible by specifying the `breakdown` argument. In the following example, estimates for each NUTS2 region are computed in addition to the overall estimate.

```
R> qsr("eqIncome", weights = "rb050", breakdown = "db040", data = eusilc)

Value:
[1] 3.971415

Value by stratum:
        stratum    value
1    Burgenland 5.073746
2     Carinthia 3.590037
3 Lower Austria 3.845026
4      Salzburg 3.829411
5        Styria 3.472333
6         Tyrol 3.628731
7 Upper Austria 3.675467
8        Vienna 4.705347
9    Vorarlberg 4.525096
```

Nevertheless, it should be noted that the quintile share ratio is highly influenced by outliers (see Hulliger and Schoch, 2009; Alfons et al., 2010). Since the upper tail of income distributions virtually always contains nonrepresentative outliers, robust estimators of the quintile share ratio should preferably be used. Thus robust semi-parametric methods based on Pareto tail modeling are implemented in package `laeken` as well. Their application is discussed in vignette `laeken-pareto` (Alfons et al., 2011c).

5.5.3 Relative median at-risk-of-poverty gap (by age and gender)

The *relative median at-risk-of-poverty gap* (RMPG) is defined as the difference between the median equivalized disposable income of persons below the at-risk-of-poverty threshold and the at-risk of poverty threshold itself, expressed as a percentage of the at-risk-of-poverty threshold (Eurostat, 2004, 2009).

For the estimation of the relative median at-risk-of-poverty gap from a sample, let $\widehat{ARPT}$ be the estimated at-risk-of-poverty threshold according to Equation (5.6), and let $I_{<\widehat{ARPT}}$ be an index set of persons with an equivalized disposable income below the estimated

at-risk-of-poverty threshold as defined in Equation (5.2). Using this index set, define $\boldsymbol{x}_{<\widehat{ARPT}} := (x_i)_{i \in I_{<\widehat{ARPT}}}$ and $\boldsymbol{w}_{<\widehat{ARPT}} := (w_i)_{i \in I_{<\widehat{ARPT}}}$. Furthermore, let $\hat{q}_{0.5}(\boldsymbol{x}_{<\widehat{ARPT}}, \boldsymbol{w}_{<\widehat{ARPT}})$ be the corresponding weighted median according to the definition in Equation (6.1). Then the relative median at-risk-of-poverty gap is estimated by

$$\widehat{RMPG} = \frac{\widehat{ARPT} - \hat{q}_{0.5}(\boldsymbol{x}_{<\widehat{ARPT}}, \boldsymbol{w}_{<\widehat{ARPT}})}{\widehat{ARPT}} \cdot 100. \tag{5.9}$$

In package `laeken`, the function `rmpg()` is implemented for the estimation of the relative median at-risk-of-poverty gap. If available in the data, sample weights should be supplied as the `weights` argument. Note that the function `arpt()` for the estimation of the at-risk-of-poverty threshold is called internally (cf. function `arpr()` for the at-risk-of-poverty rate in Section 5.5.1).

```
R> rmpg("eqIncome", weights = "rb050", data = eusilc)
```

```
Value:
[1] 18.92860

Threshold:
[1] 10859.24
```

Estimates for different subdomains can be computed by making use of the `breakdown` argument. With the following command, the overall estimate and estimates for all NUTS2 regions are computed.

```
R> rmpg("eqIncome", weights = "rb050", breakdown = "db040", data = eusilc)
```

```
Value:
[1] 18.92860

Value by stratum:
        stratum    value
1    Burgenland 12.32438
2     Carinthia 13.12787
3 Lower Austria 17.48023
4      Salzburg 28.89533
5        Styria 15.53486
6         Tyrol 19.58447
7 Upper Austria 19.47177
8        Vienna 23.35608
9    Vorarlberg 26.96706

Threshold:
[1] 10859.24
```

For the relative median at-risk-of-poverty gap, the breakdown by age and gender is of particular interest. In the following example, a breakdown variable with all possible combinations of age categories and gender is defined and added to the data set. Afterwards, estimates for the corresponding domains are computed.

```
R> ageCat <- cut(eusilc$age, c(-1, 16, 25, 50, 65, Inf), right = FALSE)
R> eusilc$breakdown <- paste(ageCat, eusilc$rb090, sep = ":")
R> rmpg("eqIncome", weights = "rb050", breakdown = "breakdown",
+     data = eusilc)

Value:
[1] 18.92860

Value by stratum:
           stratum    value
1   [-1,16):female 19.05696
2     [-1,16):male 19.05696
3   [16,25):female 32.93985
4     [16,25):male 23.70534
5   [25,50):female 20.78422
6     [25,50):male 18.19213
7   [50,65):female 21.34382
8     [50,65):male 18.92860
9  [65,Inf):female 14.48597
10   [65,Inf):male 15.34966

Threshold:
[1] 10859.24
```

5.5.4 Gini coefficient

The *Gini coefficient* is defined as the relationship of cumulative shares of the population arranged according to the level of equivalized disposable income, to the cumulative share of the equivalized total disposable income received by them (Eurostat, 2004, 2009).

For the estimation of the Gini coefficient from a sample, the sample weights need to be taken into account. In mathematical terms, the Gini coefficient is estimated by

$$\widehat{Gini} := 100 \left[\frac{2 \sum_{i=1}^{n} \left(w_i x_i \sum_{j=1}^{i} w_j \right) - \sum_{i=1}^{n} {w_i}^2 x_i}{\left(\sum_{i=1}^{n} w_i \right) \sum_{i=1}^{n} \left(w_i x_i \right)} - 1 \right]. \tag{5.10}$$

The function `gini()` is available in `laeken` to estimate the Gini coefficient. As for the other indicators, sample weights can be specified with the `weights` argument.

```
R> gini("eqIncome", weights = "rb050", data = eusilc)

Value:
[1] 26.48962
```

Using the `breakdown` argument in the following command, estimates for the NUTS2 regions are computed in addition to the overall estimate.

```
R> gini("eqIncome", weights = "rb050", breakdown = "db040", data = eusilc)
```

```
Value:
[1] 26.48962

Value by stratum:
        stratum    value
1    Burgenland 32.05489
2     Carinthia 25.49448
3 Lower Austria 25.93737
4      Salzburg 25.01652
5        Styria 23.71190
6         Tyrol 25.24881
7 Upper Austria 25.49202
8        Vienna 28.94944
9    Vorarlberg 28.74120
```

Since outliers have a strong influence on the Gini coefficient, robust estimators are preferred to the standard estimation described above (see Alfons et al., 2010). Vignette `laeken-pareto` (Alfons et al., 2011c) describes how to apply the robust semi-parametric methods implemented in package `laeken`.

5.6 Extracting information using the subset() method

If estimates of an indicator have been computed for several subdomains, it may sometimes be desired to extract the results for some domains of particular interest. In package `laeken`, this is implemented by taking advantage of the object-oriented design of the package. Each of the functions for the indicators described in Section 5.5 returns an object belonging to a class of the same name as the respective function, e.g., function `arpr()` returns an object of class `'arpr'`. All these classes thereby inherit from the basic class `'indicator'` (see Section 5.2).

```
R> a <- arpr("eqIncome", weights = "rb050", breakdown = "db040",
+     data = eusilc)
R> print(a)

Value:
[1] 14.44422

Value by stratum:
        stratum    value
1    Burgenland 19.53984
2     Carinthia 13.08627
3 Lower Austria 13.84362
4      Salzburg 13.78734
5        Styria 14.37464
6         Tyrol 15.30819
```

```
7 Upper Austria 10.88977
8        Vienna 17.23468
9    Vorarlberg 16.53731

Threshold:
[1] 10859.24
```

```
R> is.arpr(a)

[1] TRUE

R> is.indicator(a)

[1] TRUE

R> class(a)

[1] "arpr"      "indicator"
```

To extract a subset of results from such an object, a `subset()` method for the class `'indicator'` is implemented in `laeken`. The method `subset.indicator()` is hidden from the user and is called internally by the generic function `subset()` whenever an object of class `'indicator'` is supplied. In the following example, the estimates of the at-risk-of-poverty rate for the regions Lower Austria and Vienna are extracted from the object computed above.

```
R> subset(a, strata = c("Lower Austria", "Vienna"))

Value:
[1] 14.44422

Value by stratum:
        stratum    value
3 Lower Austria 13.84362
8        Vienna 17.23468

Threshold:
[1] 10859.24
```

5.7 Conclusions

This vignette demonstrates the use of package `laeken` for point estimation of the European Union indicators on social exclusion and poverty. Since the description of the indicators in EUROSTAT (2004, 2009) is weak from a mathematical point of view, a more precise notation is given in this paper. Currently, the most important indicators are implemented in `laeken`. Their estimation is made easy with the package, as it is even possible to compute estimates for several years and different subdomains with a single command.

Concerning the inequality indicators quintile share ratio and Gini coefficient, it is clearly visible from their definitions that the standard estimators are highly influenced by outliers (see also HULLIGER and SCHOCH, 2009; ALFONS et al., 2010). Therefore, robust semi-parametric methods are implemented in `laeken` as well. These are described in vignette `laeken-pareto` (ALFONS et al., 2011c), while variance and confidence interval estimation for the indicators on social exclusion and poverty with package `laeken` is treated in vignette `laeken-variance` (TEMPL and ALFONS, 2011).

Bibliography

Alfons, A., **Holzer, J.** and **Templ, M.** (**2011**a): `laeken`: Laeken indicators for measuring social cohesion. R package version 0.2.2.
URL http://CRAN.R-project.org/package=laeken

Alfons, A. and **Kraft, S.** (**2010**): `simPopulation`: Simulation of synthetic populations for surveys based on sample data. R package version 0.2.1.
URL http://CRAN.R-project.org/package=simPopulation

Alfons, A., **Kraft, S.**, **Templ, M.** and **Filzmoser, P.** (**2011**b): *Simulation of close-to-reality population data for household surveys with application to EU-SILC*. Statistical Methods & Applications, pp. 1–25, DOI 10.1007/s10260-011-0163-2.
URL http://dx.doi.org/10.1007/s10260-011-0163-2

Alfons, A., **Templ, M.**, **Filzmoser, P.** and **Holzer, J.** (**2010**): *A comparison of robust methods for Pareto tail modeling in the case of Laeken indicators*. **Borgelt, C.**, **González-Rodríguez, G.**, **Trutschnig, W.**, **Lubiano, M.**, **Gil, M.**, **Grzegorzewski, P.** and **Hryniewicz, O.** (editors) Combining Soft Computing and Statistical Methods in Data Analysis, *Advances in Intelligent and Soft Computing*, vol. 77, pp. 17–24, Heidelberg: Springer, ISBN 978-3-642-14745-6.

Alfons, A., **Templ, M.**, **Filzmoser, P.** and **Holzer, J.** (**2011**c): *Robust Pareto Tail Modeling for the Estimation of Social Inclusion Indicators using the R Package* ***laeken***. Research Report CS-2011-2, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2011-2complete.pdf

Chambers, J. and **Hastie, T.** (**1992**): Statistical Models in S. London: Chapman & Hall, ISBN 9780412830402.

Eurostat (**2004**): *Common cross-sectional EU indicators based on EU-SILC; the gender pay gap*. EU-SILC 131-rev/04, Unit D-2: Living conditions and social protection, Directorate D: Single Market, Employment and Social statistics, Eurostat, Luxembourg.

Eurostat (**2009**): *Algorithms to compute social inclusion indicators based on EU-SILC and adopted under the Open Method of Coordination (OMC)*. Doc. LC-ILC/39/09/EN-rev.1, Unit F-3: Living conditions and social protection, Directorate F: Social and information society statistics, Eurostat, Luxembourg.

Hulliger, B. and **Schoch, T.** (**2009**): *Robustification of the quintile share ratio*. New Techniques and Technologies for Statistics, Brussels.

R Development Core Team (**2011**): R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Templ, M. and **Alfons, A.** (**2011**): *Variance Estimation of Social Inclusion Indicators using the R Package* **laeken**. Research Report CS-2011-3, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2011-3complete.pdf

Chapter 6

Robust Pareto Tail Modeling with package `laeken`.

Abstract In this vignette, robust semiparametric estimation of social exclusion indicators using the R package `laeken` is discussed. Special emphasis is thereby given to income inequality indicators, as the standard estimates for these indicators are highly influenced by outliers in the upper tail of the income distribution. This influence can be reduced by modeling the upper tail with a Pareto distribution in a robust manner. While the focus of the paper is to demonstrate the functionality of `laeken` beyond the standard estimation techniques, a brief mathematical description of the implemented procedures is given as well.

6.1 Introduction

From a robustness point of view, the standard estimators for some of the social exclusion indicators defined by Eurostat (2004, 2009) are problematic. In particular the income inequality indicators *quintile share ratio* (QSR) and *Gini coefficient* suffer from a lack of robustness. Consider, e.g., the QSR, which is estimated as the ratio of estimated totals or means (see Section 6.2.1 for an exact definition). It is well known that the classical estimates for totals or means have a breakdown point of 0, meaning that even a single outlier can distort the results to an arbitrary extent. In fact, the influence of a single observation in the upper tail of the income distribution on the estimation of the QSR is linear and therefore unbounded. For practical purposes, the standard QSR estimator thus cannot be recommended in many situations (cf. Hulliger and Schoch, 2009). It is also important to note that the behavior of the Gini coefficient is similar to the behavior of the QSR.

The data basis for the estimation of the social exclusion indicators according to Eurostat (2004, 2009) is the *European Union Statistics on Income and Living Conditions* (EU-SILC), which is an annual panel survey conducted in EU member states and other European countries. On the one hand, EU-SILC data typically contain a considerable amount of *representative* outliers in the upper tail of the income distribution, i.e., correct observations that behave differently from the main part of the data, but that are not unique in the population and hence need to be considered for computing estimates of the indicators. On the other hand, EU-SILC data frequently contain some even more

extreme *nonrepresentative* outliers, i.e., observations that are either incorrect or can be considered unique in the population. Consequently, such nonrepresentative outliers need to be excluded from the estimation process or downweighted.

As a remedy, the upper tail of the income distribution may be modeled with a *Pareto distribution* in order to recalibrate the sample weights or use fitted income values for observations in the upper tail when estimating the indicators (see Section 6.6). Nevertheless, classical estimators for the parameters of the Pareto distribution are highly influenced by the nonrepresentative outliers themselves. Using robust methods reduces the influence on fitting the Pareto distribution to the representative outliers and therefore on the estimation of the indicators.

Rather than evaluating these methods, the paper concentrates on showing how they can be applied in the statistical environment R (R Development Core Team, 2011) with the add-on package `laeken` (Alfons et al., 2011a). The basic design of the package, as well as standard estimation of the social exclusion indicators is discussed in detail in vignette `laeken-standard` (Templ and Alfons, 2011a). Furthermore, the general framework for variance estimation is illustrated in vignette `laeken-variance` (Templ and Alfons, 2011b). Those documents can be viewed from within R with the following commands:

```
R> vignette("laeken-standard")
R> vignette("laeken-variance")
```

Throughout the paper, the example data from package `laeken` is used. The data set is called `eusilc` and consists of 14 827 observations from 6 000 households. In addition, it was synthetically generated from Austrian EU-SILC survey data from 2006 using the data simulation methodology proposed by Alfons et al. (2011b) and implemented in the R package `simPopulation` (Alfons and Kraft, 2010). More information on the example data can be found in vignette `laeken-standard` or in the corresponding R help page.

```
R> library("laeken")
R> data("eusilc")
```

The rest of the paper is organized as follows. Section 6.2 gives a mathematical description of the Eurostat definitions of the social exclusion indicators QSR and Gini coefficient. In Section 6.3, the Pareto distribution is briefly discussed. Section 6.4 discusses a rule of thumb for estimating the threshold for the upper tail of the distribution, and illustrates graphical methods for exploring the data in order to find the threshold. Classical and robust estimators for the shape parameter of the Pareto distribution are described in Section 6.5. How to use Pareto tail modeling to estimate the social exclusion indicators is then shown in Section 6.6. Finally, Section 7.5 concludes.

6.2 Social exclusion indicators

This paper is focused on the inequality indicators *quintile share ratio* (QSR) and *Gini coefficient*, which are both highly influenced by outliers in the upper tail of the distribution. Note that for the estimation of the social exclusion indicators, each person in a household

is assigned the same *eqivalized disposable income*. See vignette `laeken-standard` (Templ and Alfons, 2011a) for the computation of the equivalized disposable income with the R package `laeken`.

For the following definitions, let $\boldsymbol{x} := (x_1, \ldots, x_n)'$ be the equivalized disposable income with $x_1 \leq \ldots \leq x_n$ and let $\boldsymbol{w} := (w_i, \ldots, w_n)'$ be the corresponding personal sample weights, where n denotes the number of observations.

6.2.1 Quintile share ratio (QSR)

The income *quintile share ratio* (QSR) is defined as the ratio of the sum of the equivalized disposable income received by the 20% of the population with the highest equivalized disposable income to that received by the 20% of the population with the lowest equivalized disposable income (Eurostat, 2004, 2009).

For the estimation of the quintile share ratio from a sample, let $\hat{q}_{0.2}$ and $\hat{q}_{0.8}$ denote the weighted 20% and 80% quantiles, respectively. With $0 \leq p \leq 1$, these weighted quantiles are given by

$$\hat{q}_p = \hat{q}_p(\boldsymbol{x}, \boldsymbol{w}) := \begin{cases} \frac{1}{2}(x_j + x_{j+1}), & \text{if } \sum_{i=1}^{j} w_i = p \sum_{i=1}^{n} w_i, \\ x_{j+1}, & \text{if } \sum_{i=1}^{j} w_i < p \sum_{i=1}^{n} w_i < \sum_{i=1}^{j+1} w_i. \end{cases} \tag{6.1}$$

Using index sets $I_{\leq \hat{q}_{0.2}} := \{i \in \{1, \ldots, n\} : x_i \leq \hat{q}_{0.2}\}$ and $I_{> \hat{q}_{0.8}} := \{i \in \{1, \ldots, n\} : x_i > \hat{q}_{0.8}\}$, the quintile share ratio is estimated by

$$\widehat{QSR} := \frac{\sum_{i \in I_{> \hat{q}_{0.8}}} w_i x_i}{\sum_{i \in I_{\leq \hat{q}_{0.2}}} w_i x_i}. \tag{6.2}$$

With package `laeken`, the quintile share ratio can be estimated using the function `qsr()`. Sample weights can thereby be supplied via the `weights` argument.

```
R> qsr("eqIncome", weights = "rb050", data = eusilc)

Value:
[1] 3.971415
```

6.2.2 Gini coefficient

The *Gini coefficient* is defined as the relationship of cumulative shares of the population arranged according to the level of equivalized disposable income, to the cumulative share of the equivalized total disposable income received by them (Eurostat, 2004, 2009).

For the estimation of the Gini coefficient from a sample, the sample weights need to be taken into account. In mathematical terms, the Gini coefficient is estimated by

$$\widehat{Gini} := 100 \left[\frac{2 \sum_{i=1}^{n} \left(w_i x_i \sum_{j=1}^{i} w_j \right) - \sum_{i=1}^{n} {w_i}^2 x_i}{\left(\sum_{i=1}^{n} w_i \right) \sum_{i=1}^{n} \left(w_i x_i \right)} - 1 \right]. \tag{6.3}$$

The function `gini()` is available in `laeken` to estimate the Gini coefficient. As before, sample weights can be specified with the `weights` argument.

```
R> gini("eqIncome", weights = "rb050", data = eusilc)

Value:
[1] 26.48962
```

6.3 The Pareto distribution

The *Pareto distribution* is well studied in the literature and is defined in terms of its cumulative distribution function

$$F_\theta(x) = 1 - \left(\frac{x}{x_0}\right)^{-\theta}, \qquad x \geq x_0, \tag{6.4}$$

where $x_0 > 0$ is the scale parameter and $\theta > 0$ is the shape parameter (Kleiber and Kotz, 2003). Furthermore, its density function is given by

$$f_\theta(x) = \frac{\theta x_0^\theta}{x^{\theta+1}}, \qquad x \geq x_0. \tag{6.5}$$

Figure 6.1 visualizes the Pareto probability density function with scale parameter $x_0 = 1$ and different values of the shape parameter θ. Clearly, the Pareto distribution is a highly right-skewed distribution with a heavy tail. It is therefore reasonable to assume that a random variable following a Pareto distribution contains extreme values. The effect of changing the shape parameter θ is visible in the probability mass at the scale parameter x_0: the higher θ, the higher the probability mass at x_0.

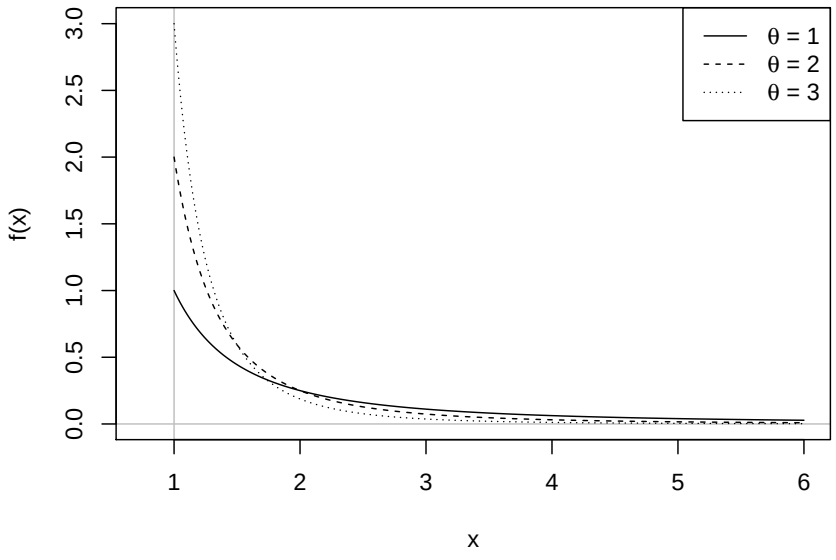

Figure 6.1: Pareto probability density functions with parameters $x_0 = 1$ and $\theta = 1, 2, 3$.

In Pareto tail modeling, the cumulative distribution function on the whole range of x is modeled as

$$F(x) = \begin{cases} G(x), & \text{if } x \leq x_0, \\ G(x_0) + (1 - G(x_0))F_\theta(x), & \text{if } x > x_0, \end{cases} \tag{6.6}$$

where G is an unknown distribution function (Dupuis and Victoria-Feser, 2006).

Let n be the number of observations and let $\boldsymbol{x} = (x_1, \ldots, x_n)'$ denote the observed values with $x_1 \leq \ldots \leq x_n$. In addition, let k be the number of observations to be used for tail modeling. In this scenario, the threshold x_0 is estimated by

$$\hat{x}_0 := x_{n-k}. \tag{6.7}$$

If an estimate $\hat{x}_0$ for the scale parameter of the Pareto distribution has been obtained, k is given by the number of observations larger than $\hat{x}_0$. Thus estimating x_0 and k directly corresponds with each other.

In the remainder of this package vignette, the equivalized disposable income of the EU-SILC example data is of main interest. Consequently, the Pareto distribution will be modeled at the household level rather than the individual level. Moreover, the focus of this vignette is on robust estimation of the social exclusion indicators. Hence the equivalized disposable income of the household with the largest income is replaced by a large outlier.

```
R> hID <- eusilc$db030[which.max(eusilc$eqIncome)]
R> eusilc[eusilc$db030 == hID, "eqIncome"] <- 1e+07
```

Since the aim is to model a Pareto distribution at the household level, the following command creates a data set that contains only the equivalized disposable income and the sample weights on the household level. This data set will be used in Sections 6.4 and 6.5 to estimate the parameters of the Pareto distribution.

```
R> eusilcH <- eusilc[!duplicated(eusilc$db030), c("eqIncome", "db090")]
```

6.4 Finding the threshold

The aim of the methods presented in this sections is to find the threshold x_0 for modeling the Pareto distribution. Several methods for the estimation of the threshold x_0 or the number of observations k in the tail have been proposed in the literature, but those proposals typically do not consider sample weights.

Beirlant et al. (1996b,a) developed a procedure that analytically determines the optimal choice of k for the Hill estimator of the shape parameter (Hill, 1975, see also Section 6.5.1 of this paper) by minimizing the asymptotic mean squared error (AMSE). In package `laeken`, this approach is implemented in the function `minAMSE()`. However, the procedure is designed for the non-robust Hill estimator and is therefore not further discussed in this paper. Furthermore, Danielsson et al. (2001) proposed a bootstrap method to find the optimal k for the Hill estimator with respect to the AMSE, which has less analytical

requirements than the approach by Beirlant et al. (1996b,a). Please note that this method is not robust either and that it is currently not available in package `laeken`. A robust prediction error criterion for choosing the number of observations k in the tail and estimating the shape parameter θ was developed by Dupuis and Victoria-Feser (2006). Nevertheless, our implementation of this robust criterion was unstable and is therefore not included in `laeken`.

In any case, Holzer (2009) concludes that graphical methods for finding the threshold outperform those analytical approaches in the case of EU-SILC data. While this section is thus focused graphical methods, a simple rule of thumb designed specifically for the equivalized disposable income in EU-SILC data is described in the following as well.

6.4.1 Van Kerm's rule of thumb

Van Kerm (2007) presented a formula that is more of a rule of thumb for the threshold of the equivalized disposable income in EU-SILC data. Is is given by

$$\hat{x}_0 := \min(\max(2.5\bar{x}, q_{0.98}), q_{0.97}), \tag{6.8}$$

where $\bar{x}$ is the weighted mean, and $q_{0.98}$ and $q_{0.97}$ are weighted quantiles as defined in Equation (6.1).

In package `laeken`, the function `paretoScale()` provides functionality for computing the threshold with van Kerm's rule of thumb. The argument `w` is available to supply sample weights.

```
R> ts <- paretoScale(eusilcH$eqIncome, w = eusilcH$db090)
R> ts

Threshold: 48459.43
Number of observations in the tail: 119
```

It should be noted that the function returns an object of class `'paretoScale'`, which consists of a component `x0` for the threshold (scale parameter) and a component `k` for the number of observations in the tail of the distribution, i.e., that are larger than the threshold.

6.4.2 Pareto quantile plot

The *Pareto quantile plot* is a graphical method for inspecting the parameters of a Pareto distribution. For the case without sample weights, it is described in detail in Beirlant et al. (1996b).

If the Pareto model holds, there exists a linear relationship between the lograrithms of the observed values and the quantiles of the standard exponential distribution, since the logarithm of a Pareto distributed random variable follows an exponential distribution. Hence the logarithms of the observed values, $\log(x_i)$, $i = 1, \ldots, n$, are plotted against the theoretical quantiles.

In the case without sample weights, the theoretical quantiles of the standard exponential distribution are given by

$$-\log\left(1-\frac{i}{n+1}\right), \qquad i=1,\ldots,n, \tag{6.9}$$

i.e., by dividing the range into $n+1$ equally sized subsets and using the resulting n inner gridpoints as probabilities for the quantiles. If the data contain sample weights, the range of the exponential distribution needs to be divided according to the weights of the n observations. The Pareto quantile plot is thus generalized by using the theoretical quantiles

$$-\log\left(1-\frac{\sum_{j=1}^{i} w_j}{\sum_{j=1}^{n} w_j}\frac{n}{n+1}\right), \qquad i=1,\ldots,n, \tag{6.10}$$

where the correction factor $\frac{n}{n+1}$ ensures that the quantiles reduce to (6.9) if all sample weights are equal.

If the tail of the data follows a Pareto distribution, those observations form almost a straight line. The leftmost point of a fitted line can thus be used as an estimate of the threshold x_0, the scale parameter. All values starting from the point after the threshold may be modeled by a Pareto distribution, but this point cannot be determined exactly. Furthermore, the slope of the fitted line is in turn an estimate of $\frac{1}{\theta}$, the reciprocal of the shape parameter.

Figure 6.2 displays the Pareto quantile plot for the example data `eusilc` on the household level with the largest observation replaced by an outlier. The plot is generated using the function `paretoQPlot()`, which allows to supply sample weights via the argument `w`. In addition, the threshold can be selected interactively by clicking on a data point. Information on the selected threshold is then printed on the R console. When the interactive selection is terminated, which is typically done by a secondary mouse click, the selected threshold is returned as an object of class `'paretoScale'`.

Another advantage of the Pareto quantile plot is also illustrated in Figure 6.2. Nonrepresentative outliers such as the large income introduced into the example data in Section 6.3, i.e., extreme observations in the upper tail that deviate from the Pareto model, are clearly visible.

6.4.3 Mean excess plot

The *mean excess plot* is another graphical method for inspecting the threshold for Pareto tail modeling, but it does not provide information on the shape parameter. It is based on the excess function

$$e(x_0) := \mathbb{E}(x-x_0|x>x_0), \qquad x_0 \geq 0. \tag{6.11}$$

A detailed description for the case without sample weights can be found in Borkovec and Klüppelberg (2000).

```
R> paretoQPlot(eusilcH$eqIncome, w = eusilcH$db090)
```

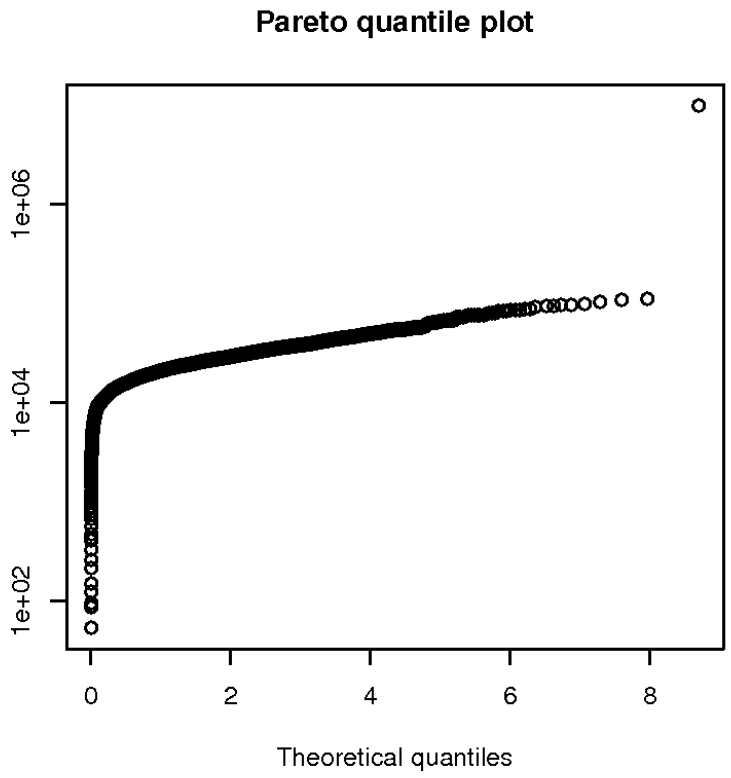

Figure 6.2: Pareto Quantile plot for the example data `eusilc` on the household level with the largest observation replaced by an outlier.

For the following definition of the mean excess plot, keep in mind that the observations are sorted such that $x_1 \leq \ldots \leq x_n$. For each observation x_i, $i = 1, \ldots, \lfloor n - \sqrt{n} \rfloor$, the empirical excess function e_n is computed. In the case without sample weights, the expectation in Equation (6.11) is replaced by the arithmetic mean, and the empirical excess function is given by

$$e_n(x_i) := \frac{1}{n-i} \sum_{j=i+1}^{n} (x_j - x_i), \qquad i = 1, \ldots, \lfloor n - \sqrt{n} \rfloor. \tag{6.12}$$

The values of the empirical excess function $e_n(x_i)$ are then plotted against the corresponding x_i, $i = 1, \ldots, \lfloor n - \sqrt{n} \rfloor$. If sample weights are available in the data, the mean excess plot is simply generalized by using the weighted mean for the empirical excess function:

$$e_n(x_i) := \frac{1}{\sum_{j=i+1}^{n} w_j} \sum_{j=i+1}^{n} w_j (x_j - x_i), \qquad i = 1, \ldots, \lfloor n - \sqrt{n} \rfloor. \tag{6.13}$$

If the tail of the data follows a Pareto distribution, those observations show a positive linear trend. The leftmost point of a fitted line can thus be used as an estimate of the

```
R> meanExcessPlot(eusilcH$eqIncome, w = eusilcH$db090)
```

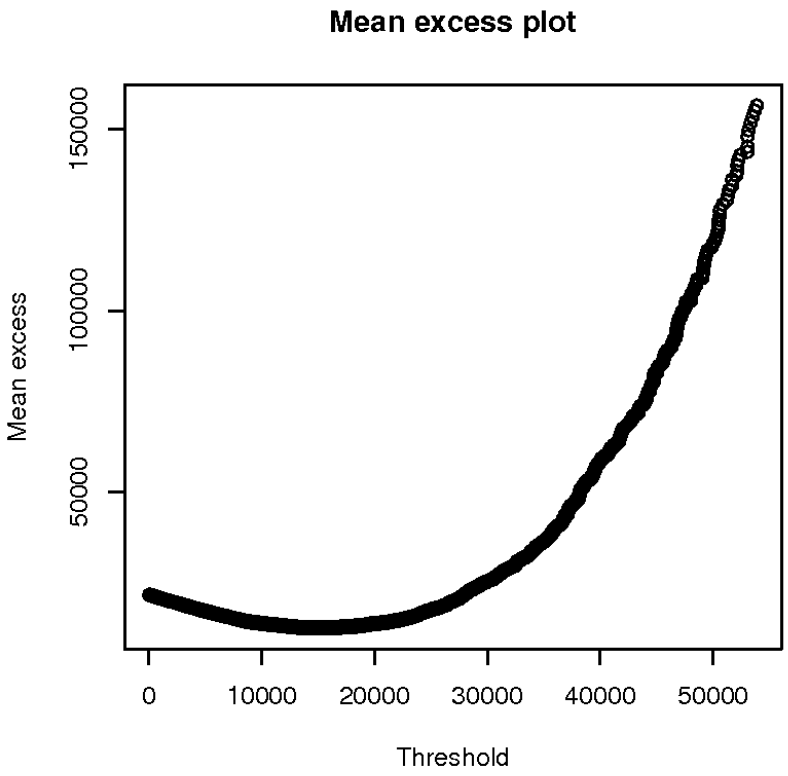

Figure 6.3: Mean excess plot for the example data `eusilc` on the household level with the largest observation replaced by an outlier.

threshold x_0, the scale parameter. As for the Pareto quantile plot, a disadvantage of the mean excess plot is that the threshold cannot be determined exactly.

Figure 6.3 shows the mean excess plot for the example data `eusilc` on the household level with the largest observation replaced by an outlier. The function `meanExcessPlot()` is thereby used to produce the plot. Sample weights can be supplied via the argument `w`. Interactive selection of the threshold works just like for the Pareto quantile plot. Again, the selected threshold is returned as an object of class `'paretoScale'`.

6.5 Estimation of the shape parameter

This section is focused on methods for estimating the shape parameter θ once the threshold x_0 is fixed. It should be noted that none of the original proposals takes sample weights into account. Most estimators presented in the following were therefore adjusted for the case of sample weights.

6.5.1 Hill estimator

The maximum likelihood estimator for the shape parameter of the Pareto distribution was introduced by Hill (1975) and is referred to as the *Hill* estimator. If the data do not contain sample weights, it is given by

$$\hat{\theta}_{\text{Hill}} = \frac{k}{\sum_{i=1}^{k} \log x_{n-k+i} - k \log x_{n-k}}. \tag{6.14}$$

In the case of sample weights, the *weighted Hill* (wHill) estimator is given by generalizing Equation (6.14) to

$$\hat{\theta}_{\text{wHill}} = \frac{\sum_{i=1}^{k} w_{n-k+i}}{\sum_{i=1}^{k} w_{n-k+i} \left(\log x_{n-k+i} - \log x_{n-k}\right)}. \tag{6.15}$$

Package `laeken` provides the function `thetaHill()` to compute the Hill estimator. It requires to specify either the number of observations in the tail via the argument `k`, or the threshold via the argument `x0`. Furthermore, the argument `w` can be used to supply sample weights. In the following example, the shape parameter is estimated using the largest observations (first command) and the threshold (second command) as computed with van Kerm's rule of thumb in Section 6.4.1.

```
R> thetaHill(eusilcH$eqIncome, k = ts$k, w = eusilcH$db090)

[1] 3.437979

R> thetaHill(eusilcH$eqIncome, x0 = ts$x0, w = eusilcH$db090)

[1] 3.437979
```

6.5.2 Weighted maximum likelihood estimator

The *weighted maximum likelihood* (WML) estimator (Dupuis and Morgenthaler, 2002; Dupuis and Victoria-Feser, 2006) falls into the class of M-estimators and is given by the solution $\hat{\theta}$ of

$$\sum_{i=1}^{k} \Psi(x_{n-k+i}, \theta) = 0 \tag{6.16}$$

with

$$\Psi(x, \theta) := u(x, \theta) \frac{\partial}{\partial \theta} \log f(x, \theta) = u(x, \theta) \left(\frac{1}{\theta} - \log \frac{x}{x_0} \right), \tag{6.17}$$

where $u(x, \theta)$ is a weight function with values in $[0, 1]$. In the implementation in package `laeken`, a Huber type weight function is used by default, as proposed by Dupuis and Victoria-Feser (2006). Let the logarithms of the relative excesses be denoted by

$$z_i := \log \left(\frac{x_{n-k+i}}{x_{n-k}} \right), \qquad i = 1, \ldots, k. \tag{6.18}$$

In the Pareto model, these can be predicted by

$$\hat{z}_i := -\frac{1}{\theta} \log\left(\frac{k+1-i}{k+1}\right), \qquad i = 1, \ldots, k. \tag{6.19}$$

The variance of z_i is given by

$${\sigma_i}^2 := \sum_{j=1}^{i} \frac{1}{\theta^2 (k-i+j)^2}, \qquad i = 1, \ldots, k. \tag{6.20}$$

Using the standardized residuals

$$r_i := \frac{z_i - \hat{z}_i}{\sigma_i}, \tag{6.21}$$

the Huber type weight function with tuning constant c is defined as

$$u(x_{n-k+i}, \theta) := \begin{cases} 1, & \text{if } |r_i| \leq c, \\ \frac{c}{|r_i|}, & \text{if } |r_i| > c. \end{cases} \tag{6.22}$$

For this choice of weight function, the bias of $\hat{\theta}$ is approximated by

$$\hat{B}(\hat{\theta}) = -\frac{\sum_{i=1}^{k} \left(u_i \frac{\partial}{\partial\theta} \log f_i\right)|_{\hat{\theta}} \left(F_{\hat{\theta}}(x_{n-k+i}) - F_{\hat{\theta}}(x_{n-k+i-1})\right)}{\sum_{i=1}^{k} \left(\frac{\partial}{\partial\theta} u_i \frac{\partial}{\partial\theta} \log f_i + u_i \frac{\partial^2}{\partial\theta^2} \log f_i\right)|_{\hat{\theta}} \left(F_{\hat{\theta}}(x_{n-k+i}) - F_{\hat{\theta}}(x_{n-k+i-1})\right)}, \tag{6.23}$$

where $u_i := u(x_{n-k+i}, \theta)$ and $f_i := f(x_{n-k+i}, \theta)$. This term is used to obtain a bias-corrected estimator

$$\tilde{\theta} := \hat{\theta} - \hat{B}(\hat{\theta}). \tag{6.24}$$

For details and proofs of the above statements, as well as for information on a probability-based weight function $u(x, \theta)$, the reader is referred to Dupuis and Morgenthaler (2002) and Dupuis and Victoria-Feser (2006). However, note the WML estimator does not consider sample weights. An adjustment of the estimator to take sample weights into account is currently not available due to its complexity. For sampling designs that lead to equal sample weights, the WML estimator may still be useful, though.

The function `thetaWML()` is available in `laeken` to compute the WML estimator. Again, either the argument `k` or `x0` needs to be used to specify the number of observations in the tail or the threshold. Since the sample weights in the example data are not equal, the following example is only included to demonstrate the use of the function.

```
R> thetaWML(eusilcH$eqIncome, k = ts$k)

[1] 4.226204

R> thetaWML(eusilcH$eqIncome, x0 = ts$x0)

[1] 4.226204
```

6.5.3 Integrated squared error estimator

For the *integrated squared error* (ISE) estimator (Vandewalle et al., 2007), the Pareto distribution is modeled in terms of the relative excesses

$$y_i := \frac{x_{n-k+i}}{x_{n-k}}, \qquad i = 1, \ldots, k. \tag{6.25}$$

The density function of the Pareto distribution for the relative excesses is approximated by

$$f_\theta(y) = \theta y^{-(1+\theta)}. \tag{6.26}$$

The ISE estimator is then given by minimizing the integrated squared error criterion (Terrell, 1990):

$$\hat{\theta} = \arg\min_\theta \left[\int f_\theta^2(y)dy - 2\mathbb{E}(f_\theta(Y)) \right]. \tag{6.27}$$

If there are no sample weights in the data, the mean is used as an unbiased estimator of $\mathbb{E}(f_\theta(Y))$ in order to obtain the ISE estimate

$$\hat{\theta}_{\text{ISE}} = \arg\min_\theta \left[\int f_\theta^2(y)dy - \frac{2}{k}\sum_{i=1}^{k} f_\theta(y_i) \right]. \tag{6.28}$$

See Vandewalle et al. (2007) for more information on the ISE estimator for the case without sample weights.

If sample weights are available in the data, the mean in Equation (6.28) is simply replaced by a weighted mean to obtain the *weighted integrated squared error* (wISE) estimator:

$$\hat{\theta}_{\text{wISE}} = \arg\min_\theta \left[\int f_\theta^2(y)dy - \frac{2}{\sum_{i=1}^{k} w_{n-k+i}} \sum_{i=1}^{k} w_{n-k+i} f_\theta(y_i) \right]. \tag{6.29}$$

With package `laeken`, the ISE estimator can be computed using the function `thetaISE()`. The arguments `k` and `x0` are available to specify either the number of observations in the tail or the threshold, and sample weights can be supplied via the argument `w`.

```
R> thetaISE(eusilcH$eqIncome, k = ts$k, w = eusilcH$db090)

[1] 3.993801

R> thetaISE(eusilcH$eqIncome, x0 = ts$x0, w = eusilcH$db090)

[1] 3.993801
```

6.5.4 Partial density component estimator

For the *partial density component* (PDC) estimator Vandewalle et al. (2007) minimizes the integrated squared error criterion using an incomplete density mixture model uf_θ. If the data do not contain sample weights, the PDC estimator in is thus given by

$$\hat{\theta}_{\text{PDC}} = \arg\min_{\theta} \left[u^2 \int f_\theta^2(y)dy - \frac{2u}{k} \sum_{i=1}^{k} f_\theta(y_i) \right]. \tag{6.30}$$

The parameter u can be interpreted as a measure of the uncontaminated part of the sample and is estimated by

$$\hat{u} = \frac{\frac{1}{k}\sum_{i=1}^{k} f_{\hat{\theta}}(y_i)}{\int f_{\hat{\theta}}^2(y)dy}. \tag{6.31}$$

See Vandewalle et al. (2007) and references therein for more information on the PDC estimator for the case without sample weights.

Taking sample weights into account, the *weighted partial density component* (wPDC) estimator is obtained by generalizing Equations (6.30) and (6.31) to

$$\hat{\theta}_{\text{wPDC}} = \arg\min_{\theta} \left[u^2 \int f_\theta^2(y)dy - \frac{2u}{\sum_{i=1}^{k} w_{n-k+i}} \sum_{i=1}^{k} w_{n-k+i} f_\theta(y_i) \right], \tag{6.32}$$

$$\hat{u} = \frac{\frac{1}{\sum_{i=1}^{k} w_{n-k+i}} \sum_{i=1}^{k} w_{n-k+i} f_{\hat{\theta}}(y_i)}{\int f_{\hat{\theta}}^2(y)dy}. \tag{6.33}$$

The function `thetaPDC()` is implemented in package `laeken` to compute the PDC estimator. As for the other estimators, it is necessary to specify either the number of observations in the tail via the argument `k`, or the threshold via the argument `x0`. Sample weights can be supplied using the argument `w`.

```
R> thetaPDC(eusilcH$eqIncome, k = ts$k, w = eusilcH$db090)

[1] 4.132596

R> thetaPDC(eusilcH$eqIncome, x0 = ts$x0, w = eusilcH$db090)

[1] 4.132596
```

6.6 Estimation of the indicators using Pareto tail modeling

Three approaches based on Pareto tail modeling for reducing the influence of outliers on the social exclusion indicators are implemented in the R package `laeken`:

Calibration for nonrepresentative outliers (CN): Values larger than a certain quantile of the fitted distribution are declared as nonrepresentative outliers. Since these are considered to be unique to the population data, the sample weights of the corresponding observations are set to 1 and the weights of the remaining observations are adjusted accordingly by calibration.

Replacement of nonrepresentative outliers (RN): Values larger than a certain quantile of the fitted distribution are declared as nonrepresentative outliers. Only these nonrepresentative outliers are replaced by values drawn from the fitted distribution, thereby preserving the order of the original values.

Replacement of the tail (RT): All values above the threshold are replaced by values drawn from the fitted distribution. The order of the original values is preserved.

An evaluation of the RT approach by means of a simulation study can be found in Alfons et al. (2010).

Keep in mind that the largest observation in the example data `eusilc` was replaced by a large outlier in Section 6.3. With the following command, the Gini coefficient is estimated according to the Eurostat definition to show that even a single outlier can completely distort the results for the standard estimation (see Section 6.2.2 for the original value).

```
R> gini("eqIncome", weights = "rb050", data = eusilc)

Value:
[1] 29.24333
```

For Pareto tail modeling, the function `paretoTail()` is implemented in `laeken`. It returns an object of class `'paretoTail'`, which contains all the necessary information for further analysis using the three approaches described above. Note that the household IDs are supplied via the argument `groups` such that the Pareto distribution is fitted on the household level rather than the individual level. In addition, the PDC is used by default to estimate the shape parameter. Other estimators can be specified via the `method` argument.

```
R> fit <- paretoTail(eusilc$eqIncome, k = ts$k, w = eusilc$db090,
+     groups = eusilc$db030)
```

The function `reweightOut()` is available for semiparametric estimation with the CN approach. It returns a vector of the recalibrated weights. In this example, regional information is used as auxiliary variables for calibration. The function `calibVars()` thereby transforms a factor into a matrix of binary variables, as required by the calibration function `calibWeights()`, which is called internally. These recalibrated weights are then simply used to estimate the Gini coefficient with function `gini()`.

```
R> w <- reweightOut(fit, calibVars(eusilc$db040))
R> gini(eusilc$eqIncome, w)

Value:
[1] 26.45973
```

For the RN approach, the function `replaceOut()` is implemented. Since values are drawn from the fitted distribution to replace the observations flagged as outliers, the seed of the random number generator is set first for reproducibility of the results. The returned vector of incomes is then supplied to `gini()` to estimate the Gini coefficient.

```
R> set.seed(1234)
R> eqIncome <- replaceOut(fit)
R> gini(eqIncome, weights = eusilc$rb050)

Value:
[1] 26.46924
```

Similarly, the function `replaceTail()` is available for the RT approach. Again, the seed of the random number generator is set beforehand.

```
R> set.seed(1234)
R> eqIncome <- replaceTail(fit)
R> gini(eqIncome, weights = eusilc$rb050)

Value:
[1] 26.64921
```

It should be noted that `replaceTail()` can also be used for the RN approach by setting the argument `all` to `FALSE`. In fact, `replaceOut(x, ...)` is a simple wrapper for `replaceTail(x, all = FALSE, ...)`.

In any case, the estimates for the semiparametric approaches based on Pareto tail modeling are very close to the original value before the outlier has been introduced (see Section 6.2.2), whereas the standard estimation is corrupted by the outlier. Furthermore, the estimation of other indicators such as the quintile share ratio (see Section 6.2.1) using the semiparametric approaches is straightforward and hence not shown here.

6.7 Conclusions

This vignette shows the functionality of package `laeken` for robust semiparametric estimation of social exclusion indicators based on Pareto tail modeling. Most notably, it demonstrates that the functions are easy to use and that the implementation follows an object-oriented design. While the focus of the paper lies on the use of the package, a mathematical description of the methods is given as well.

Furthermore, it is shown that the standard estimation of the inequality indicators can be corrupted by a single outlier, thus underlining the need for robust alternatives. Three approaches for robust semiparametric estimation based on Pareto tail modeling are thereby implemented such that the corresponding functions share a common interface for ease of use.

Bibliography

Alfons, A., **Holzer, J.** and **Templ, M.** (**2011**a): `laeken`: Laeken indicators for measuring social cohesion. R package version 0.2.2.
URL http://CRAN.R-project.org/package=laeken

Alfons, A. and **Kraft, S.** (**2010**): `simPopulation`: Simulation of synthetic populations for surveys based on sample data. R package version 0.2.1.
URL http://CRAN.R-project.org/package=simPopulation

Alfons, A., **Kraft, S.**, **Templ, M.** and **Filzmoser, P.** (**2011**b): *Simulation of close-to-reality population data for household surveys with application to EU-SILC*. Statistical Methods & Applications, pp. 1–25, DOI 10.1007/s10260-011-0163-2.
URL http://dx.doi.org/10.1007/s10260-011-0163-2

Alfons, A., **Templ, M.**, **Filzmoser, P.** and **Holzer, J.** (**2010**): *A comparison of robust methods for Pareto tail modeling in the case of Laeken indicators*. **Borgelt, C.**, **González-Rodríguez, G.**, **Trutschnig, W.**, **Lubiano, M.**, **Gil, M.**, **Grzegorzewski, P.** and **Hryniewicz, O.** (editors) Combining Soft Computing and Statistical Methods in Data Analysis, *Advances in Intelligent and Soft Computing*, vol. 77, pp. 17–24, Heidelberg: Springer, ISBN 978-3-642-14745-6.

Beirlant, J., **Vynckier, P.** and **Teugels, J.** (**1996**a): *Excess functions and estimation of the extreme-value index*. Bernoulli, 2 (4), pp. 293–318.

Beirlant, J., **Vynckier, P.** and **Teugels, J.** (**1996**b): *Tail index estimation, Pareto quantile plots, and regression diagnostics*. Journal of the American Statistical Association, 31 (436), pp. 1659–1667.

Borkovec, M. and **Klüppelberg, C.** (**2000**): *Extremwerttheorie für Finanzzeitreihen – ein unverzichtbares Werkzeug im Risikomanagement*. **Johanning, L.** and **Rudolph, B.** (editors) Handbuch Risikomanagement, pp. 219–241, Bad Soden: Uhlenbruch, ISBN 3933207150.

Danielsson, J., **de Haan, L.**, **Peng, L.** and **de Vries, C.** (**2001**): *Using a bootstrap method to choose the sample fraction in tail index estimation*. Journal of Multivariate Analysis, 76 (2), pp. 226–248.

Dupuis, D. and **Morgenthaler, S.** (**2002**): *Robust weighted likelihood estimators with an application to bivariate extreme value problems*. The Canadian Journal of Statistics, 30 (1), pp. 17–36.

Dupuis, D. and **Victoria-Feser, M.-P.** (**2006**): *A robust prediction error criterion for Pareto modelling of upper tails*. The Canadian Journal of Statistics, 34 (4), pp. 639–658.

Eurostat (**2004**): *Common cross-sectional EU indicators based on EU-SILC; the gender pay gap*. EU-SILC 131-rev/04, Unit D-2: Living conditions and social protection, Directorate D: Single Market, Employment and Social statistics, Eurostat, Luxembourg.

Eurostat (**2009**): *Algorithms to compute social inclusion indicators based on EU-SILC and adopted under the Open Method of Coordination (OMC)*. Doc. LC-ILC/39/09/EN-rev.1, Unit F-3: Living conditions and social protection, Directorate F: Social and information society statistics, Eurostat, Luxembourg.

Hill, B. (**1975**): *A simple general approach to inference about the tail of a distribution*. The Annals of Statistics, 3 (5), pp. 1163–1174.

Holzer, J. (**2009**): Robust methods for the estimation of selected Laeken indicators. Diploma thesis, Department of Statistics and Probability Theory, Vienna University of Technology, Vienna, Austria.

Hulliger, B. and **Schoch, T.** (**2009**): *Robustification of the quintile share ratio*. New Techniques and Technologies for Statistics, Brussels.

Kleiber, C. and **Kotz, S.** (**2003**): Statistical Size Distributions in Economics and Actuarial Sciences. Hoboken: John Wiley & Sons, ISBN 0-471-15064-9.

R Development Core Team (**2011**): R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Templ, M. and **Alfons, A.** (**2011**a): *Standard Methods for Point Estimation of Social Inclusion Indicators using the R Package* ***laeken***. Research Report CS-2011-1, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2011-1complete.pdf

Templ, M. and **Alfons, A.** (**2011**b): *Variance Estimation of Social Inclusion Indicators using the R Package* ***laeken***. Research Report CS-2011-3, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2011-3complete.pdf

Terrell, G. (**1990**): *Linear density estimates*. Proceedings of the Statistical Computing Section, pp. 297–302, American Statistical Association.

Van Kerm, P. (**2007**): *Extreme incomes and the estimation of poverty and inequality indicators from EU-SILC*. IRISS Working Paper Series 2007-01, CEPS/INSTEAD.

Vandewalle, B., **Beirlant, J.**, **Christmann, A.** and **Hubert, M.** (**2007**): *A robust estimator for the tail index of Pareto-type distributions*. Computational Statistics & Data Analysis, 51 (12), pp. 6252–6268.

Chapter 7

Variance Estimation of Indicators using package `laeken`

Abstract This vignette illustrates the application of variance estimation procedures to indicators on social exclusion and poverty using the R package `laeken`. To be more precise, it describes a general framework for estimating variance and confidence intervals of indicators under complex sampling designs. Currently, the package is focused on bootstrap approaches. While the naive bootstrap does not modify the weights of the bootstrap samples, a calibrated version allows to calibrate each bootstrap sample on auxiliary information before deriving the bootstrap replicate estimate.

Keywords: Variance estimation, software, R

7.1 Introduction

When point estimates of indicators are computed from samples, it is important to also obtain variance estimates and confidence intervals in order to account for variability due to sampling. Other sources of variability such as data editing or imputation may need to be considered as well, but this is not further discussed in this paper. While this vignette targets the topic of variance and confidence interval estimation for the indicators on social exclusion and poverty according to Eurostat (2004, 2009), the aim is not to describe and evaluate the different approaches that have been proposed to date. Instead, the aim is to present the functionality for the statistical environment R (R Development Core Team, 2011) implemented in the add-on package `laeken` (Alfons et al., 2011a).

It should be noted that the basic design of the package, as well as standard point estimation of the indicators on social exclusion and poverty, is discussed in detail in vignette `laeken-standard` (Templ and Alfons, 2011). In addition, vignette `laeken-pareto` (Alfons et al., 2011c) presents more sophisticated methods for point estimation of the indicators, which are less influenced by outliers. Those documents can be viewed from within R with the following commands:

```
R> vignette("laeken-standard")
R> vignette("laeken-pareto")
```

The data basis for the estimation of the indicators on social exclusion and poverty is the *European Union Statistics on Income and Living Conditions* (EU-SILC), which is an annual panel survey conducted in EU member states and other European countries. Package `laeken` provides the synthetic example data `eusilc` consisting of 14 827 observations from 6 000 households. Furthermore, the data were generated from Austrian EU-SILC survey data from 2006 using the data simulation methodology proposed by Alfons et al. (2011b) and implemented in the R package `simPopulation` (Alfons and Kraft, 2010). The data set `eusilc` is used in the code examples throughout the paper.

```
R> library("laeken")
R> data("eusilc")
```

The rest of the paper is organized as follows. Section 7.2 presents the general wrapper function for estimating variance and confidence intervals of indicators in package `laeken`. The naive and calibrated bootstrap approaches are discussed in Sections 7.3 and 7.4, respectively. Section 7.5 concludes.

7.2 General wrapper function for variance estimation

The function `variance()` provides a flexible framework for estimating the variance and confidence intervals of indicators such as the *at-risk-of-poverty rate*, the *Gini coefficient*, the *quintile share ratio* and the *relative median at-risk-of-poverty gap*. For a mathematical description and details on the implementation of these indicators in the R package `laeken`, the reader is referred to vignette `laeken-standard` (Templ and Alfons, 2011). In any case, `variance()` acts as a general wrapper function for computing variance and confidence interval estimates of indicators on social exclusion and poverty with package `laeken`. The arguments of function `variance()` are shown in the following:

```
R> args(variance)
```

```
function (inc, weights = NULL, years = NULL, breakdown = NULL,
    design = NULL, data = NULL, indicator, alpha = 0.05, na.rm = FALSE,
    type = "bootstrap", gender = NULL, method = "mean", ...)
NULL
```

All these arguments are fully described in the R help page of function `variance()`. The most important arguments are:

inc: the income vector.

weights: an optional vector of sample weights.

breakdown: an optional vector giving different domains in which variances and confidence intervals should be computed.

design: an optional vector or factor giving different strata for stratified sampling designs.

data: an optional `data.frame`. If supplied, each of the above arguments should be specified as a character string or an integer or logical vector specifying the corresponding column.

indicator: an object inheriting from the class `'indicator'` that contains the point estimates of the indicator, such as `'arpr'` for the at-risk-of-poverty rate, `'qsr'` for the quintile share ratio, `'rmpg'` for the relative median at-risk-of-poverty gap, or `'gini'` for the Gini coefficient.

type: a character string specifying the type of variance estimation to be used. Currently, only `'bootstrap'` is implemented for variance estimation based on bootstrap resampling.

In the following sections, two bootstrap methods for estimating the variance and confidence intervals of point estimates for complex survey data are described. Furthermore, their application using the function `variance()` from package `laeken` is demonstrated.

7.3 Naive bootstrap

Let $\boldsymbol{X} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)'$ denote a survey sample with n observations and p variables. Then the *naive bootstrap algorithm* for estimating the variance and confidence interval of an indicator can be summarized as follows:

1. Draw R independent bootstrap samples $\boldsymbol{X}_1^*, \ldots, \boldsymbol{X}_R^*$ from $\boldsymbol{X}$.
2. Compute the bootstrap replicate estimates $\hat{\theta}_r^* := \hat{\theta}(\boldsymbol{X}_r^*)$ for each bootstrap sample $\boldsymbol{X}_r^*$, $r = 1, \ldots, R$, where $\hat{\theta}$ denotes an estimator for a certain indicator of interest. Of course the sample weights always need to be considered for the computation of the bootstrap replicate estimates.
3. Estimate the variance $V(\hat{\theta})$ by the variance of the R bootstrap replicate estimates:

$$\hat{V}(\hat{\theta}) := \frac{1}{R-1} \sum_{r=1}^{R} \left(\hat{\theta}_r^* - \frac{1}{R} \sum_{s=1}^{R} \hat{\theta}_s^* \right)^2 . \tag{7.1}$$

4. Estimate the confidence interval at confidence level $1 - \alpha$ by one of the following methods (for details, see DAVISON and HINKLEY, 1997):

   Percentile method: $\left[\hat{\theta}^*_{((R+1)\frac{\alpha}{2})}, \hat{\theta}^*_{((R+1)(1-\frac{\alpha}{2}))} \right]$, as suggested by EFRON and TIBSHIRANI (1993).

   Normal approximation: $\hat{\theta} \pm z_{1-\frac{\alpha}{2}} \cdot \hat{V}(\hat{\theta})^{1/2}$ with $z_{1-\frac{\alpha}{2}} = \Phi^{-1}(1 - \frac{\alpha}{2})$.

   Basic bootstrap method: $\left[2\hat{\theta} - \hat{\theta}^*_{((R+1)(1-\frac{\alpha}{2}))}, 2\hat{\theta} - \hat{\theta}^*_{((R+1)\frac{\alpha}{2})} \right]$.

   For the percentile and the basic bootstrap method, $\hat{\theta}^*_{(1)} \leq \ldots \leq \hat{\theta}^*_{(R)}$ denote the order statistics of the bootstrap replicate estimates.

In the following example, the variance and confidence interval of the at-risk-of-poverty rate are estimated with the naive bootstrap procedure. The output of function `variance()` is an object of the same class as the point estimate supplied as the `indicator` argument, but with additional components for the variance and confidence interval. In addition to the

point estimate, the income and the sample weights need to be supplied. Furthermore, a stratified sampling design can be considered by specifying the `design` argument, in which case observations are resampled separately within the strata. To ensure reproducibility of the results, the seed of the random number generator is set.

```
R> a <- arpr("eqIncome", weights = "rb050", data = eusilc)
R> variance("eqIncome", weights = "rb050", design = "db040", data = eusilc,
+     indicator = a, bootType = "naive", seed = 123)

Value:
[1] 14.44422

Variance:
[1] 0.0920564

Confidence interval:
   lower    upper
13.87663 15.19417

Threshold:
[1] 10859.24
```

One of the most convenient features of package `laeken` is that indicators can be evaluated for different subdomains using a single command. This also holds for variance estimation. Using the `breakdown` argument, the example below produces variance and confidence interval estimates for each NUTS2 region in addition to the overall values.

```
R> b <- arpr("eqIncome", weights = "rb050", breakdown = "db040",
+     data = eusilc)
R> variance("eqIncome", weights = "rb050", breakdown = "db040",
+     design = "db040", data = eusilc, indicator = b, bootType = "naive",
+     seed = 123)

Value:
[1] 14.44422

Variance:
[1] 0.0920564

Confidence interval:
   lower    upper
13.87663 15.19417

Value by stratum:
        stratum    value
1    Burgenland 19.53984
2     Carinthia 13.08627
3 Lower Austria 13.84362
4      Salzburg 13.78734
```

```
5        Styria 14.37464
6         Tyrol 15.30819
7 Upper Austria 10.88977
8        Vienna 17.23468
9    Vorarlberg 16.53731

Variance by stratum:
        stratum       var
1    Burgenland 3.5105237
2     Carinthia 1.4133369
3 Lower Austria 0.4456053
4      Salzburg 1.2937926
5        Styria 0.4615967
6         Tyrol 1.0299617
7 Upper Austria 0.3785766
8        Vienna 0.6384621
9    Vorarlberg 1.7601223

Confidence interval by stratum:
        stratum     lower    upper
1    Burgenland 16.072806 23.63099
2     Carinthia 10.640776 15.23716
3 Lower Austria 12.196265 15.17182
4      Salzburg 11.913708 16.13315
5        Styria 13.020339 15.89730
6         Tyrol 13.084487 17.51124
7 Upper Austria  9.960467 12.54200
8        Vienna 15.712609 18.96003
9    Vorarlberg 13.604720 19.23431

Threshold:
[1] 10859.24
```

It should be noted that the workhorse function `bootVar()` is called internally by `variance()` for bootstrap variance and confidence interval estimation. The function `bootVar()` could also be called directly by the user in exactly the same manner. Moreover, variance and confidence interval estimation for any other indicator implemented in package `laeken` is straightforward—the application using function `variance()` or `bootVar()` remains the same.

7.4 Calibrated bootstrap

Rao and Wu (1988) showed that the naive bootstrap is biased when used in the complex survey context. They propose to increase the variance estimate in the h-th stratum by a factor of $\frac{n_h - 1}{n_h}$ (if the bootstrap sample is of the same size). In addition, they describe

extensions to sampling without replacement, unequal probability sampling, and two-stage cluster sampling with equal probabilities and without replacement.

Deville and Särndal (1992) and Deville et al. (1993) provide a general description on how to calibrate sample weights to account for known population totals. The naive bootstrap does not include the recalibration of bootstrap samples in order to fit known population totals and therefore is, strictly formulated, not suitable for many practical applications. However, even though a bias might be introduced, the naive bootstrap works well in many situations and is faster to compute than the calibrated version. Hence it is a popular method often used in practice.

In real-world data, the inclusion probabilities for observations in the population are in general not all equal, resulting in different *design weights* for the observations in the sample. Furthermore, the initial design weights are in practice often adjusted by calibration, e.g., to account for non-response or so that certain known population totals can be precisely estimated from the survey sample. To give a simplified example, if the population sizes in different regions are known, the sample weights may be calibrated so that the Horvitz-Thompson estimates (Horvitz and Thompson, 1952) of the population sizes equal the known true values. However, when bootstrap samples are drawn from survey data, resampling observations has the effect that such known population totals can no longer be precisely estimated. As a remedy, the sample weights of each bootstrap sample should be calibrated.

The calibrated version of the bootstrap thus results in more precise variance and confidence interval estimation, but comes with higher computational costs than the naive approach. In any case, the *calibrated bootstrap algorithm* is obtained by adding the following step between Steps 1 and 2 of the naive bootstrap algorithm from Section 7.3:

1b. Calibrate the sample weights for each bootstrap sample $\boldsymbol{X}_r^*$, $r = 1, \ldots, R$. Generalized raking procedures are thereby used for calibration: either a multiplicative method known as *raking*, an additive method or a logit method (see Deville and Särndal, 1992; Deville et al., 1993).

The function call to `variance()` for the calibrated bootstrap is very similar to its counterpart for the naive bootstrap. A matrix of auxiliary calibration variables needs to be supplied via the argument `aux`. In addition, the argument `totals` can be used to supply the corresponding population totals. If the `totals` argument is omitted, as in the following example, the population totals are computed from the sample weights of the original sample. This follows the assumption that those weights are already calibrated on the supplied auxiliary variables.

```
R> variance("eqIncome", weights = "rb050", design = "db040", data = eusilc,
+     indicator = a, X = calibVars(eusilc$db040), seed = 123)

Value:
[1] 14.44422

Variance:
[1] 0.09165169
```

```
Confidence interval:
   lower    upper
13.87817 15.19303

Threshold:
[1] 10859.24
```

Note that the function `calibVars()` transforms a factor into a matrix of binary variables, as required by the calibration function `calibWeights()`, which is called internally. While the default is to use raking for calibration, other methods can be specified via the `method` argument.

7.5 Conclusions

Both bootstrap procedures for variance and confidence interval estimation of indicators on social exclusion and poverty currently implemented in the R package `laeken` have their strengths. While the naive bootstrap is faster to compute, the calibrated bootstrap in general leads to more precise results. The implementation of other procedures such as linearization techniques (Kovačević and Binder, 1997; Deville, 1999; Hulliger and Münnich, 2006; Osier, 2009) or the delete-a-group jackknife (Kott, 2001) is future work.

Furthermore, Alfons et al. (2009) demonstrated how the variance of indicators computed from data with imputed values may be underestimated in bootstrap procedures, depending on the indicator itself and the imputation procedure used. They proposed to use the method described in Little and Rubin (2002), which consists of drawing bootstrap samples from the original data with missing values, and to impute the missing data for each bootstrap sample before computing the corresponding bootstrap replicate estimate. Of course, this results in an additional increase of the computation time. The implementation of this procedure in package `laeken` is future work. It should also be noted that multiple imputation is a further possibility to consider the additional uncertainty from imputation when estimating the variance of an indicator (see Little and Rubin, 2002).

Bibliography

Alfons, A., **Holzer, J.** and **Templ, M.** (**2011**a): `laeken`: Laeken indicators for measuring social cohesion. R package version 0.2.2.
URL http://CRAN.R-project.org/package=laeken

Alfons, A. and **Kraft, S.** (**2010**): `simPopulation`: Simulation of synthetic populations for surveys based on sample data. R package version 0.2.1.
URL http://CRAN.R-project.org/package=simPopulation

Alfons, A., **Kraft, S.**, **Templ, M.** and **Filzmoser, P.** (**2011**b): *Simulation of close-to-reality population data for household surveys with application to EU-SILC*. Statistical

Methods & Applications, pp. 1–25, DOI 10.1007/s10260-011-0163-2.
URL http://dx.doi.org/10.1007/s10260-011-0163-2

Alfons, A., **Templ, M.** and **Filzmoser, P.** (**2009**): *On the influence of imputation methods on Laeken indicators: simulations and recommendations*. UNECE Worksession on Statistical Data Editing, Neuchâtel, Switzerland.

Alfons, A., **Templ, M.**, **Filzmoser, P.** and **Holzer, J.** (**2011**c): *Robust Pareto Tail Modeling for the Estimation of Social Inclusion Indicators using the R Package laeken*. Research Report CS-2011-2, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2011-2complete.pdf

Davison, A. and **Hinkley, D.** (**1997**): Bootstrap Methods and their Applications. Cambridge University Press, ISBN 0 521 57471 4.

Deville, J.-C. (**1999**): *Variance estimation for complex statistics and estimators: Linearization and residual techniques*. Survey Methodology, 25 (2), pp. 193–203.

Deville, J.-C. and **Särndal, C.-E.** (**1992**): *Calibration estimators in survey sampling*. Journal of the American Statistical Association, 87 (418), pp. 376–382.

Deville, J.-C., **Särndal, C.-E.** and **Sautory, O.** (**1993**): *Generalized raking procedures in survey sampling*. Journal of the American Statistical Association, 88 (423), pp. 1013–1020.

Efron, B. and **Tibshirani, R.** (**1993**): An Introduction to the Bootstrap. New York: Chapman & Hall, ISBN 0-412-04231-2.

Eurostat (**2004**): *Common cross-sectional EU indicators based on EU-SILC; the gender pay gap*. EU-SILC 131-rev/04, Unit D-2: Living conditions and social protection, Directorate D: Single Market, Employment and Social statistics, Eurostat, Luxembourg.

Eurostat (**2009**): *Algorithms to compute social inclusion indicators based on EU-SILC and adopted under the Open Method of Coordination (OMC)*. Doc. LC-ILC/39/09/EN-rev.1, Unit F-3: Living conditions and social protection, Directorate F: Social and information society statistics, Eurostat, Luxembourg.

Horvitz, D. and **Thompson, D.** (**1952**): *A generalization of sampling without replacement from a finite universe*. Journal of the American Statistical Association, 47 (260), pp. 663–685.

Hulliger, B. and **Münnich, R.** (**2006**): *Variance estimation for complex surveys in the presence of outliers*. Proceedings of the Section on Survey Research Methods, pp. 3153–3156, American Statistical Association.

Kott, P. (**2001**): *The delete-a-group jackknife*. Journal of Official Statistics, 17 (4), pp. 521–526.

Kovačević, M. and **Binder, D.** (**1997**): *Variance estimation for measures for income inequality and polarization – The estimating equations approach*. Journal of Official Statistics, 13 (1), pp. 41–58.

Little, R. and **Rubin, D.** (**2002**): Statistical Analysis with Missing Data. New York: John Wiley & Sons, 2nd ed., ISBN 0-471-18386-5.

Osier, G. (**2009**): *Variance estimation for complex indicators of poverty and inequality using linearization techniques*. Survey Research Methods, 3 (3), pp. 167–195.

R Development Core Team (**2011**): R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.
URL http://www.R-project.org

Rao, J. and **Wu, C.** (**1988**): *Resampling inference with complex survey data*. Journal of the American Statistical Association, 83 (401), pp. 231–241.

Templ, M. and **Alfons, A.** (**2011**): *Standard Methods for Point Estimation of Social Inclusion Indicators using the R Package* **laeken**. Research Report CS-2011-1, Department of Statistics and Probability Theory, Vienna University of Technology.
URL http://www.statistik.tuwien.ac.at/forschung/CS/CS-2011-1complete.pdf

Package 'simFrame'

March 21, 2011

Version 0.4.1

Date 2011-03-21

Title Simulation framework

Author Andreas Alfons

Maintainer Andreas Alfons <alfons@statistik.tuwien.ac.at>

Depends R (>= 2.10.0), Rcpp (>= 0.8.6), lattice, snow

Imports lattice, methods, stats, stats4, utils

Suggests laeken, mvtnorm, robCompositions, sampling

LinkingTo Rcpp

SystemRequirements GNU make

Description A general framework for statistical simulation.

License GPL (>= 2)

LazyLoad yes

Repository CRAN

Date/Publication 2011-03-21 18:23:42

R topics documented:

simFrame-package . . . 3
accessors . . . 5
aggregate-methods . . . 12
BasicVector-class . . . 14
clusterRunSimulation . . . 16
clusterSetup . . . 20
contaminate . . . 23
ContControl . . . 25

ContControl-class . . . 27
DARContControl-class . . . 28
DataControl-class . . . 30
DCARContControl-class . . . 32
draw . . . 34
eusilcP . . . 36
generate . . . 38
head-methods . . . 39
inclusionProb . . . 41
length-methods . . . 42
NAControl-class . . . 43
NumericMatrix-class . . . 45
OptBasicVector-class . . . 46
OptCall-class . . . 47
OptCharacter-class . . . 47
OptContControl-class . . . 48
OptDataControl-class . . . 49
OptNAControl-class . . . 49
OptNumeric-class . . . 50
OptSampleControl-class . . . 51
plot-methods . . . 51
runSimulation . . . 53
SampleControl-class . . . 57
SampleSetup-class . . . 59
sampling . . . 61
setNA . . . 63
setup . . . 65
simApply . . . 67
simBwplot . . . 69
SimControl-class . . . 71
simDensityplot . . . 75
SimResults-class . . . 77
simSample . . . 80
simXyplot . . . 81
Strata-class . . . 84
stratify . . . 86
stratify-utilities . . . 87
summary-methods . . . 89
SummarySampleSetup-class . . . 90
tail-methods . . . 92
TwoStageControl-class . . . 94
VirtualContControl-class . . . 97
VirtualDataControl-class . . . 98
VirtualNAControl-class . . . 99
VirtualSampleControl-class . . . 101

Index **103**

`simFrame-package` *Simulation framework*

Description

A general framework for statistical simulation.

Details

Package:	simFrame
Version:	0.4.1
Date:	2011-03-21
Depends:	R (>= 2.10.0), Rcpp (>= 0.8.6), lattice, snow
Imports:	lattice, methods, stats, stats4, utils
Suggests:	laeken, mvtnorm, robCompositions, sampling
LinkingTo:	Rcpp
SystemRequirements:	GNU make
License:	GPL (>= 2)
LazyLoad:	yes

Index:


```
BasicVector-class       Class "BasicVector"
ContControl             Create contamination control objects
ContControl-class       Class "ContControl"
DARContControl-class    Class "DARContControl"
DCARContControl-class   Class "DCARContControl"
DataControl-class       Class "DataControl"
NAControl-class         Class "NAControl"
NumericMatrix-class     Class "NumericMatrix"
OptBasicVector-class    Class "OptBasicVector"
OptCall-class           Class "OptCall"
OptCharacter-class      Class "OptCharacter"
OptContControl-class    Class "OptContControl"
OptDataControl-class    Class "OptDataControl"
OptNAControl-class      Class "OptNAControl"
OptNumeric-class        Class "OptNumeric"
OptSampleControl-class
                        Class "OptSampleControl"
SampleControl-class     Class "SampleControl"
SampleSetup-class       Class "SampleSetup"
SimControl-class        Class "SimControl"
SimResults-class        Class "SimResults"
Strata-class            Class "Strata"
SummarySampleSetup-class
```

```
                          Class "SummarySampleSetup"
TwoStageControl-class     Class "TwoStageControl"
VirtualContControl-class
                          Class "VirtualContControl"
VirtualDataControl-class
                          Class "VirtualDataControl"
VirtualNAControl-class
                          Class "VirtualNAControl"
VirtualSampleControl-class
                          Class "VirtualSampleControl"
aggregate-methods         Method for aggregating simulation results
clusterRunSimulation      Run a simulation experiment on a snow cluster
clusterSetup              Set up multiple samples on a snow cluster
contaminate               Contaminate data
draw                      Draw a sample
eusilcP                   Synthetic EU-SILC data
generate                  Generate data
getAdd                    Accessor and mutator functions for objects
getStrataLegend           Utility functions for stratifying data
head-methods              Methods for returning the first parts of an
                          object
inclusionProb             Inclusion probabilities
length-methods            Methods for getting the length of an object
plot-methods              Plot simulation results
runSimulation             Run a simulation experiment
setNA                     Set missing values
setup                     Set up multiple samples
simApply                  Apply a function to subsets
simBwplot                 Box-and-whisker plots
simDensityplot            Kernel density plots
simFrame-package          Simulation framework
simSample                 Set up multiple samples
simXyplot                 X-Y plots
srs                       Random sampling
stratify                  Stratify data
summary-methods           Methods for producing a summary of an object
tail-methods              Methods for returning the last parts of an
                          object
```

Further information is available in the following vignettes:

simFrame-eusilc	Applications of Statistical Simulation in the Case of EU-SILC: Using the R Package simFrame (sourc
simFrame-intro	An Object-Oriented Framework for Statistical Simulation: The R Package simFrame (source, pdf)

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* simFrame.

Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

Maintainer: Andreas Alfons, <alfons@statistik.tuwien.ac.at>

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

`accessors` *Accessor and mutator functions for objects*

Description

Get values of slots of objects via accessor functions and set values via mutator functions. If no mutator methods are available, the slots of the corresponding objects are not supposed to be changed by the user.

Usage

```
getAdd(x)

getAux(x)
setAux(x, aux)

getCall(x)

getCollect(x)
setCollect(x, collect)

getColnames(x)
setColnames(x, colnames)

getContControl(x)
setContControl(x, contControl)

getControl(x)

getDataControl(x)

getDesign(x)
setDesign(x, design)
```

```
getDistribution(x)
setDistribution(x, distribution)

getDots(x, ...)
setDots(x, dots, ...)

## S4 method for signature 'TwoStageControl'
getDots(x, stage = NULL)
## S4 method for signature 'TwoStageControl'
setDots(x, dots, stage = NULL)

getEpsilon(x)
setEpsilon(x, epsilon)

getFun(x, ...)
setFun(x, fun, ...)

## S4 method for signature 'TwoStageControl'
getFun(x, stage = NULL)
## S4 method for signature 'TwoStageControl'
setFun(x, fun, stage = NULL)

getGrouping(x)
setGrouping(x, grouping)

getIndices(x)

getIntoContamination(x)
setIntoContamination(x, intoContamination)

getK(x)
setK(x, k)

getLegend(x)

getNAControl(x)
setNAControl(x, NAControl)

getNArate(x)
setNArate(x, NArate)

getNr(x)

getNrep(x)

getProb(x, ...)
setProb(x, prob, ...)
```

```
## S4 method for signature 'TwoStageControl'
getProb(x, stage = NULL)
## S4 method for signature 'TwoStageControl'
setProb(x, prob, stage = NULL)

getSAE(x)
setSAE(x, SAE)

getSampleControl(x)

getSeed(x)

getSize(x, ...)
setSize(x, size, ...)

## S4 method for signature 'TwoStageControl'
getSize(x, stage = NULL)
## S4 method for signature 'TwoStageControl'
setSize(x, size, stage = NULL)

getSplit(x)

getTarget(x)
setTarget(x, target)

getValues(x)
```

Arguments

x	an object.
aux	a character string specifying an auxiliary variable (see "ContControl" and "NAControl").
collect	a logical indicating whether groups should be collected after sampling individuals or sampled directly (see "SampleControl").
colnames	a character vector specifying column names (see "DataControl").
contControl	an object of class "ContControl" (see "SimControl").
design	a character vector specifying columns to be used for stratification (see "SampleControl", "TwoStageControl" and "SimControl").
distribution	a function generating data (see "DataControl" and "DCARContControl").
dots	additional arguments to be passed to a function (see "DataControl", "DARContControl", "DCARContControl", "SampleControl", "TwoStageControl" and "SimControl").
epsilon	a numeric vector giving contamination levels (see "VirtualContControl").
fun	a function (see "DARContControl", "SampleControl", "TwoStageControl" and "SimControl").

grouping	a character string specifying a grouping variable (see "ContControl", "NAControl", "SampleControl" and "TwoStageControl").
intoContamination	a logical indicating whether missing values should also be inserted into contaminated observations (see "NAControl").
k	a single positive integer giving the number of samples to be set up (see "VirtualSampleControl").
NAControl	an object of class "NAControl" (see "SimControl").
NArate	a numeric vector or matrix giving missing value rates (see "VirtualNAControl").
prob	a numeric vector giving probability weights (see "SampleControl" and "TwoStageControl").
SAE	a logical indicating whether small area estimation will be used in the simulation experiment (see "SimControl").
size	a non-negative integer or a vector of non-negative integers (see "DataControl", "SampleControl" and "TwoStageControl").
stage	optional integer; for certain slots of "TwoStageControl", this allows to access or modify only the list component for the specified stage. Use 1 for the first stage and 2 for the second stage.
target	a character vector specifying target columns (see "VirtualContControl" and "VirtualNAControl").
...	only used to allow for the stage argument in accessor and mutator methods for "TwoStageControl". Otherwise no additional arguments are available.

Value

For accessor functions, the corresponding slot of x is returned.

For mutator functions, the corresponding slot of x is replaced.

Methods for function getAdd

```
signature(x = "SimResults")
```

Methods for functions getAux and setAux

```
signature(x = "ContControl")
signature(x = "NAControl")
```

Methods for function getCall

```
signature(x = "SampleSetup")
signature(x = "SimResults")
signature(x = "Strata")
```

Methods for functions getCollect and setCollect

```
signature(x = "SampleControl")
```

accessors 9

Methods for function getColnames

```
signature(x = "DataControl")
signature(x = "SimResults")
```

Methods for function setColnames

```
signature(x = "DataControl")
```

Methods for functions getContControl and setContControl

```
signature(x = "SimControl")
```

Methods for function getControl

```
signature(x = "SampleSetup")
signature(x = "SimResults")
```

Methods for function getDataControl

```
signature(x = "SimResults")
```

Methods for function getDesign

```
signature(x = "SampleControl")
signature(x = "TwoStageControl")
signature(x = "SimControl")
signature(x = "SimResults")
signature(x = "Strata")
```

Methods for function setDesign

```
signature(x = "SampleControl")
signature(x = "TwoStageControl")
signature(x = "SimControl")
```

Methods for functions getDistribution and setDistribution

```
signature(x = "DataControl")
signature(x = "DCARContControl")
```

Methods for functions getDots and setDots

```
signature(x = "DataControl")
signature(x = "DARContControl")
signature(x = "DCARContControl")
signature(x = "SampleControl")
signature(x = "TwoStageControl")
signature(x = "SimControl")
```

Methods for function getEpsilon

```
signature(x = "SimResults")
signature(x = "VirtualContControl")
```

Methods for function setEpsilon

```
signature(x = "VirtualContControl")
```

Methods for functions getFun and setFun

```
signature(x = "DARContControl")
signature(x = "SampleControl")
signature(x = "TwoStageControl")
signature(x = "SimControl")
```

Methods for functions getGrouping and setGrouping

```
signature(x = "ContControl")
signature(x = "NAControl")
signature(x = "SampleControl")
signature(x = "TwoStageControl")
```

Methods for function getIndices

```
signature(x = "SampleSetup")
```

Methods for functions getIntoContamination and setIntoContamination

```
signature(x = "NAControl")
```

Methods for functions getK and setK

```
signature(x = "VirtualSampleControl")
```

Methods for function getLegend

```
signature(x = "Strata")
```

Methods for functions getNAControl and setNAControl

```
signature(x = "SimControl")
```

Methods for function getNArate

```
signature(x = "SimResults")
signature(x = "VirtualNAControl")
```

accessors 11

Methods for function setNArate

```
signature(x = "VirtualNAControl")
```

Methods for function getNr

```
signature(x = "Strata")
```

Methods for function getNrep

```
signature(x = "SimResults")
```

Methods for function getProb

```
signature(x = "SampleControl")
signature(x = "TwoStageControl")
signature(x = "SampleSetup")
```

Methods for function setProb

```
signature(x = "SampleControl")
signature(x = "TwoStageControl")
```

Methods for functions getSAE and setSAE

```
signature(x = "SimControl")
```

Methods for function getSampleControl

```
signature(x = "SimResults")
```

Methods for function getSeed

```
signature(x = "SampleSetup")
signature(x = "SimResults")
```

Methods for function getSize

```
signature(x = "DataControl")
signature(x = "SampleControl")
signature(x = "TwoStageControl")
signature(x = "Strata")
signature(x = "SummarySampleSetup")
```

Methods for function setSize

```
signature(x = "DataControl")
signature(x = "SampleControl")
signature(x = "TwoStageControl")
```

Methods for function getSplit

```
signature(x = "Strata")
```

Methods for functions getTarget and setTarget

```
signature(x = "VirtualContControl")
signature(x = "VirtualNAControl")
```

Methods for function getValues

```
signature(x = "SimResults")
signature(x = "Strata")
```

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

Examples

```
nc <- NAControl(NArate = 0.05)
getNArate(nc)

setNArate(nc, c(0.01, 0.03, 0.05, 0.07, 0.09))
getNArate(nc)
```

---

aggregate-methods *Method for aggregating simulation results*

Description

Aggregate simulation results, i.e, split the data into subsets if applicable and compute summary statistics.

Usage

```
## S4 method for signature 'SimResults'
aggregate(x, select = NULL, FUN = mean, ...)
```

Arguments

`x`	the simulation results to be aggregated, i.e., an object of class `"SimResults"`.
`select`	a character vector specifying the columns to be aggregated. It must be a subset of the `colnames` slot of `x`, which is the default.
`FUN`	a scalar function to compute the summary statistics (defaults to `mean`).
`...`	additional arguments to be passed down to `aggregate` or `apply`.

Value

If contamination or missing values have been inserted or the simulations have been split into different domains, a `data.frame` is returned, otherwise a vector.

Details

If contamination or missing values have been inserted or the simulations have been split into different domains, `aggregate` is called to compute the summary statistics for the respective subsets.

Otherwise, `apply` is called to compute the summary statistics for each column specified by `select`.

Methods

`x = "SimResults"` aggregate simulation results.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`aggregate`, `apply`, `"SimResults"`

Examples

```
#### design-based simulation
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome", epsilon = 0.02,
    fun = function(x) x * 25)

## function for simulation runs
sim <- function(x) {
```

```
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
}

## run simulation
results <- runSimulation(eusilcP,
    sc, contControl = cc, fun = sim)

## aggregate
aggregate(results)  # means of results
aggregate(results, FUN = sd)  # standard deviations of results


#### model-based simulation
set.seed(12345)  # for reproducibility

## function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}

## run simulation
results <- runSimulation(dc, nrep = 50,
    contControl = cc, design = "group", fun = sim)

## aggregate
aggregate(results)  # means of results
aggregate(results, FUN = sd)  # standard deviations of results
```

`BasicVector-class` *Class "BasicVector"*

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "OptBasicVector", directly.

Methods

`getStrataLegend signature(x = "data.frame", design = "BasicVector")`: get a `data.frame` describing the strata.

`getStrataSplit signature(x = "data.frame", design = "BasicVector")`: get a list in which each element contains the indices of the observations belonging to the corresponding stratum.

`getStrataTable signature(x = "data.frame", design = "BasicVector")`: get a `data.frame` describing the strata and containing the stratum sizes.

`getStratumSizes signature(x = "data.frame", design = "BasicVector")`: get the stratum sizes.

`getStratumValues signature(x = "data.frame", design = "BasicVector", split = "missing")`: get the stratum number for each observation.

`getStratumValues signature(x = "data.frame", design = "BasicVector", split = "list")`: get the stratum number for each observation.

`simApply signature(x = "data.frame", design = "BasicVector", fun = "function")`: apply a function to subsets.

`simSapply signature(x = "data.frame", design = "BasicVector", fun = "function")`: apply a function to subsets.

`stratify signature(x = "data.frame", design = "BasicVector")`: stratify data.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

Examples

```
showClass("BasicVector")
```

clusterRunSimulation
Run a simulation experiment on a snow cluster

Description

Generic function for running a simulation experiment on a `snow` cluster.

Usage

```
clusterRunSimulation(cl, x, setup, nrep, control,
                     contControl = NULL, NAControl = NULL,
                     design = character(), fun, ...,
                     SAE = FALSE)
```

Arguments

`cl`	a `snow` cluster.
`x`	a `data.frame` (for design-based simulation or simulation based on real data) or a control object for data generation inheriting from `"VirtualDataControl"` (for model-based simulation or mixed simulation designs).
`setup`	an object of class `"SampleSetup"`, containing previously set up samples, or a control class for setting up samples inheriting from `"VirtualSampleControl"`.
`nrep`	a non-negative integer giving the number of repetitions of the simulation experiment (for model-based simulation, mixed simulation designs or simulation based on real data).
`control`	a control object of class `"SimControl"`
`contControl`	an object of a class inheriting from `"VirtualContControl"`, controlling contamination in the simulation experiment.
`NAControl`	an object of a class inheriting from `"VirtualNAControl"`, controlling the insertion of missing values in the simulation experiment.
`design`	a character vector specifying variables (columns) to be used for splitting the data into domains. The simulations, including contamination and the insertion of missing values (unless `SAE=TRUE`), are then performed on every domain.
`fun`	a function to be applied in each simulation run.
`...`	for `runSimulation`, additional arguments to be passed to `fun`. For `runSim`, arguments to be passed to `runSimulation`.
`SAE`	a logical indicating whether small area estimation will be used in the simulation experiment.

Details

Statistical simulation is embarrassingly parallel, hence computational performance can be increased by parallel computing. In `simFrame`, parallel computing is implemented using the package `snow`. Note that all objects and packages required for the computations (including `simFrame`) need to be made available on every worker process.

In order to prevent problems with random numbers and to ensure reproducibility, random number streams should be used. In R, the packages `rlecuyer` and `rsprng` are available for creating random number streams, which are supported by `snow` via the function `clusterSetupRNG`.

There are some requirements for slot `fun` of the control object `control`. The function must return a numeric vector, or a list with the two components `values` (a numeric vector) and `add` (additional results of any class, e.g., statistical models). Note that the latter is computationally slightly more expensive. A `data.frame` is passed to `fun` in every simulation run. The corresponding argument must be called `x`. If comparisons with the original data need to be made, e.g., for evaluating the quality of imputation methods, the function should have an argument called `orig`. If different domains are used in the simulation, the indices of the current domain can be passed to the function via an argument called `domain`.

For small area estimation, the following points have to be kept in mind. The slot `design` of `control` for splitting the data must be supplied and the slot `SAE` must be set to `TRUE`. However, the data are not actually split into the specified domains. Instead, the whole data set (sample) is passed to `fun`. Also contamination and missing values are added to the whole data (sample). Last, but not least, the function must have a `domain` argument so that the current domain can be extracted from the whole data (sample).

In every simulation run, `fun` is evaluated using `try`. Hence no results are lost if computations fail in any of the simulation runs.

Value

An object of class `"SimResults"`.

Methods

`cl = "ANY", x = "ANY", setup = "ANY", nrep = "ANY", control = "missing"`
convenience wrapper that allows the slots of `control` to be supplied as arguments

`cl = "ANY", x = "data.frame", setup = "missing", nrep = "numeric", control = "SimCo`
run a simulation experiment based on real data with repetitions on a `snow` cluster.

`cl = "ANY", x = "data.frame", setup = "SampleSetup", nrep = "missing", control = "S`
run a design-based simulation experiment with previously set up samples on a `snow` cluster.

`cl = "ANY", x = "data.frame", setup = "VirtualSampleControl", nrep = "missing", cor`
run a design-based simulation experiment on a `snow` cluster.

`cl = "ANY", x = "VirtualDataControl", setup = "missing", nrep = "numeric", control`
run a model-based simulation experiment with repetitions on a `snow` cluster.

`cl = "ANY", x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numer`
run a simulation experiment using a mixed simulation design with repetitions on a `snow` cluster.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

L'Ecuyer, P., Simard, R., Chen E and Kelton, W. (2002) An Object-Oriented Random-Number Package with Many Long Streams and Substreams. *Operations Research*, **50**(6), 1073–1075.

Mascagni, M. and Srinivasan, A. (2000) Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation. *ACM Transactions on Mathematical Software*, **26**(3), 436–461.

Rossini, A., Tierney L. and Li, N. (2007) Simple Parallel Statistical Computing in R. *Journal of Computational and Graphical Statistics*, **16**(2), 399–420.

Tierney, L., Rossini, A. and Li, N. (2009) snow: A Parallel Computing Framework for the R System. *International Journal of Parallel Programming*, **37**(1), 78–90.

See Also

makeCluster, clusterSetupRNG, runSimulation, "SimControl", "SimResults", simBwplot, simDensityplot, simXyplot

Examples

```
## Not run:
## these examples requires at least dual core processor


## design-based simulation
data(eusilcP)  #load data

# start snow cluster
cl <- makeCluster(2, type = "SOCK")

# load package and data on workers
clusterEvalQ(cl, {
    library(simFrame)
    data(eusilcP)
})

# set up random number stream
clusterSetupRNG(cl, seed = "12345")

# control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome", epsilon = 0.02,
    fun = function(x) x * 25)

# function for simulation runs
```

```
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
}

# export objects to workers
clusterExport(cl, c("sc", "cc", "sim"))

# run simulation on snow cluster
results <- clusterRunSimulation(cl, eusilcP,
    sc, contControl = cc, fun = sim)

# stop snow cluster
stopCluster(cl)

# explore results
head(results)
aggregate(results)
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)



## model-based simulation

# start snow cluster
cl <- makeCluster(2, type = "SOCK")

# load package on workers
clusterEvalQ(cl, library(simFrame))

# set up random number stream
clusterSetupRNG(cl, seed = "12345")

# function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

# control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

# function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}
```

```
# export objects to workers
clusterExport(cl, c("rgnorm", "means", "dc", "cc", "sim"))

# run simulation on snow cluster
results <- clusterRunSimulation(cl, dc, nrep = 100,
    contControl = cc, design = "group", fun = sim)

# stop snow cluster
stopCluster(cl)

# explore results
head(results)
aggregate(results)
plot(results, true = means)

## End(Not run)
```

`clusterSetup` *Set up multiple samples on a snow cluster*

Description

Generic function for setting up multiple samples on a `snow` cluster.

Usage

```
clusterSetup(cl, x, control, ...)

## S4 method for signature 'ANY,data.frame,SampleControl'
clusterSetup(cl, x, control)
```

Arguments

`cl`	a `snow` cluster.
`x`	the `data.frame` to sample from.
`control`	a control object inheriting from the virtual class `"VirtualSampleControl"` or a character string specifying such a control class (the default being `"SampleControl"`).
`...`	if `control` is a character string or missing, the slots of the control object may be supplied as additional arguments. See `"SampleControl"` for details on the slots.

Details

A fundamental design principle of the framework in the case of design-based simulation studies is that the sampling procedure is separated from the simulation procedure. Two main advantages arise from setting up all samples in advance.

First, the repeated sampling reduces overall computation time dramatically in certain situations, since computer-intensive tasks like stratification need to be performed only once. This is particularly relevant for large population data. In close-to-reality simulation studies carried out in research projects in survey statistics, often up to 10000 samples are drawn from a population of millions of individuals with stratified sampling designs. For such large data sets, stratification takes a considerable amount of time and is a very memory-intensive task. If the samples are taken on-the-fly, i.e., in every simulation run one sample is drawn, the function to take the stratified sample would typically split the population into the different strata in each of the 10000 simulation runs. If all samples are drawn in advance, on the other hand, the population data need to be split only once and all 10000 samples can be taken from the respective strata together.

Second, the samples can be stored permanently, which simplifies the reproduction of simulation results and may help to maximize comparability of results obtained by different partners in a research project. In particular, this is useful for large population data, when complex sampling techniques may be very time-consuming. In research projects involving different partners, usually different groups investigate different kinds of estimators. If the two groups use not only the same population data, but also the same previously set up samples, their results are highly comparable.

The computational performance of setting up multiple samples can be increased by parallel computing. In `simFrame`, parallel computing is implemented using the package `snow`. Note that all objects and packages required for the computations (including `simFrame`) need to be made available on every worker process.

In order to prevent problems with random numbers and to ensure reproducibility, random number streams should be used. In R, the packages `rlecuyer` and `rsprng` are available for creating random number streams, which are supported by `snow` via the function `clusterSetupRNG`.

The control class `"SampleControl"` is highly flexible and allows stratified sampling as well as sampling of whole groups rather than individuals with a specified sampling method. Hence it is often sufficient to implement the desired sampling method for the simple non-stratified case to extend the existing framework. See `"SampleControl"` for some restrictions on the argument names of such a function, which should return a vector containing the indices of the sampled observations.

Nevertheless, for very complex sampling procedures, it is possible to define a control class `"MySampleControl"` extending `"VirtualSampleControl"`, and the corresponding method `clusterSetup(cl, x, control)` with signature `'ANY, data.frame, MySampleControl'`. In order to optimize computational performance, it is necessary to efficiently set up multiple samples. Thereby the slot `k` of `"VirtualSampleControl"` needs to be used to control the number of samples, and the resulting object must be of class `"SampleSetup"`.

Value

An object of class `"SampleSetup"`.

Methods

`cl = "ANY", x = "data.frame", control = "character"` set up multiple samples on a `snow` cluster using a control class specified by the character string `control`. The slots of the control object may be supplied as additional arguments.

`cl = "ANY", x = "data.frame", control = "missing"` set up multiple samples on a `snow` cluster using a control object of class `"SampleControl"`. Its slots may be supplied as additional arguments.

`cl = "ANY", x = "data.frame", control = "SampleControl"` set up multiple samples on a `snow` cluster as defined by the control object `control`.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

L'Ecuyer, P., Simard, R., Chen E and Kelton, W. (2002) An Object-Oriented Random-Number Package with Many Long Streams and Substreams. *Operations Research*, **50**(6), 1073–1075.

Mascagni, M. and Srinivasan, A. (2000) Algorithm 806: `SPRNG`: A Scalable Library for Pseudorandom Number Generation. *ACM Transactions on Mathematical Software*, **26**(3), 436–461.

Rossini, A., Tierney L. and Li, N. (2007) Simple Parallel Statistical Computing in R. *Journal of Computational and Graphical Statistics*, **16**(2), 399–420.

Tierney, L., Rossini, A. and Li, N. (2009) `snow`: A Parallel Computing Framework for the R System. *International Journal of Parallel Programming*, **37**(1), 78–90.

See Also

`makeCluster`, `clusterSetupRNG`, `setup`, `draw`, `"SampleControl"`, `"TwoStageControl"`, `"VirtualSampleControl"`, `"SampleSetup"`

Examples

```
## Not run:
# these examples require at least dual core processor

# load data
data(eusilcP)

# start snow cluster
cl <- makeCluster(2, type = "SOCK")

# load package and data on workers
clusterEvalQ(cl, {
        library(simFrame)
        data(eusilcP)
    })

# simple random sampling
srss <- clusterSetup(cl, eusilcP, size = 20, k = 4)
summary(srss)
draw(eusilcP[, c("id", "eqIncome")], srss, i = 1)

# group sampling
gss <- clusterSetup(cl, eusilcP, grouping = "hid", size = 10, k = 4)
```

```
summary(gss)
draw(eusilcP[, c("hid", "id", "eqIncome")], gss, i = 2)

# stratified simple random sampling
ssrss <- clusterSetup(cl, eusilcP, design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(ssrss)
draw(eusilcP[, c("id", "region", "eqIncome")], ssrss, i = 3)

# stratified group sampling
sgss <- clusterSetup(cl, eusilcP, design = "region",
    grouping = "hid", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(sgss)
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgss, i = 4)

# stop snow cluster
stopCluster(cl)

## End(Not run)
```

`contaminate` *Contaminate data*

Description

Generic function for contaminating data.

Usage

```
contaminate(x, control, ...)

## S4 method for signature 'data.frame,ContControl'
contaminate(x, control, i)
```

Arguments

`x`	the data to be contaminated.
`control`	a control object of a class inheriting from the virtual class `"VirtualContControl"` or a character string specifying such a control class (the default being `"DCARContControl"`).
`i`	an integer giving the element of the slot `epsilon` of `control` to be used as contamination level.
`...`	if `control` is a character string or missing, the slots of the control object may be supplied as additional arguments. See `"DCARContControl"` and `"DARContControl"` for details on the slots.

Details

With the control classes implemented in **simFrame**, contamination is modeled as a two-step process. The first step is to select observations to be contaminated, the second is to model the distribution of the outliers.

In order to extend the framework by a user-defined control class `"MyContControl"` (which must extend `"VirtualContControl"`), a method `contaminate(x, control, i)` with signature `'data.frame, MyContControl'` needs to be implemented. In case the contaminated observations need to be identified at a later stage of the simulation, e.g., if conflicts with inserting missing values should be avoided, a logical indicator variable `".contaminated"` should be added to the returned data set.

Value

A `data.frame` containing the contaminated data. In addition, the column `".contaminated"`, which consists of logicals indicating the contaminated observations, is added to the `data.frame`.

Methods

`x = "data.frame", control = "character"` contaminate data using a control class specified by the character string `control`. The slots of the control object may be supplied as additional arguments.

`x = "data.frame", control = "ContControl"` contaminate data as defined by the control object `control`.

`x = "data.frame", control = "missing"` contaminate data using a control object of class `"ContControl"`. Its slots may be supplied as additional arguments.

Note

Since version 0.3, `contaminate` no longer checks if the auxiliary variable with probability weights are numeric and contain only finite positive values (`sample` still throws an error in these cases). This has been removed to improve computational performance in simulation studies.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

Alfons, A., Templ, M. and Filzmoser, P. (2010) Contamination Models in the R Package **simFrame** for Statistical Simulation. In Aivazian, S., Filzmoser, P. and Kharin, Y. (editors) *Computer Data Analysis and Modeling: Complex Stochastic Data and Systems*, volume 2, 178–181. Minsk. ISBN 978-985-476-848-9.

Béguin, C. and Hulliger, B. (2008) The BACON-EEM Algorithm for Multivariate Outlier Detection in Incomplete Survey Data. *Survey Methodology*, **34**(1), 91–103.

Hulliger, B. and Schoch, T. (2009) Robust Multivariate Imputation with Survey Data. *57th Session of the International Statistical Institute*, Durban.

See Also

"DCARContControl", "DARContControl", "ContControl", "VirtualContControl"

Examples

```
## distributed completely at random
data(eusilcP)
sam <- draw(eusilcP[, c("id", "eqIncome")], size = 20)

# using a control object
dcarc <- ContControl(target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000), type = "DCAR")
contaminate(sam, dcarc)

# supply slots of control object as arguments
contaminate(sam, target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000))


## distributed at random
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
foo <- generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))

# using a control object
darc <- DARContControl(target = "V2",
    epsilon = 0.2, fun = function(x) x * 100)
contaminate(foo, darc)

# supply slots of control object as arguments
contaminate(foo, "DARContControl", target = "V2",
    epsilon = 0.2, fun = function(x) x * 100)
```

`ContControl` *Create contamination control objects*

Description

Create objects of a class inheriting from "ContControl".

Usage

```
ContControl(..., type = c("DCAR", "DAR"))
```

Arguments

`...`	arguments passed to `new("DCARContControl", ...)` or `new("DARContControl", ...)`, as determined by `type`.
`type`	a character string specifying whether a control object of class `"DCARContControl"` or `"DARContControl"` should be created.

Value

If `type = "DCAR"`, an object of class `"DCARContControl"`.

If `type = "DAR"`, an object of class `"DARContControl"`.

Note

This constructor exists mainly for back compatibility with early draft versions of `simFrame`.

Author(s)

Andreas Alfons

See Also

`"DCARContControl"`, `"DARContControl"`, `"ContControl"`

Examples

```
## distributed completely at random
data(eusilcP)
sam <- draw(eusilcP[, c("id", "eqIncome")], size = 20)
dcarc <- ContControl(target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000), type = "DCAR")
contaminate(sam, dcarc)

## distributed at random
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
foo <- generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
darc <- ContControl(target = "V2", epsilon = 0.2,
    fun = function(x) x * 100, type = "DAR")
contaminate(foo, darc)
```

`ContControl-class` *Class "ContControl"*

Description

Virtual class for controlling contamination in a simulation experiment (used internally).

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`target`: Object of class `"OptCharacter"`; a character vector specifying specifying the variables (columns) to be contaminated, or `NULL` to contaminate all variables (except the additional ones generated internally).

`epsilon`: Object of class `"numeric"` giving the contamination levels.

`grouping`: Object of class `"character"` specifying a grouping variable (column) to be used for contaminating whole groups rather than individual observations.

`aux`: Object of class `"character"` specifying an auxiliary variable (column) whose values are used as probability weights for selecting the items (observations or groups) to be contaminated.

Extends

Class `"VirtualContControl"`, directly. Class `"OptContControl"`, by class "VirtualContControl", distance 2.

Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from `"VirtualContControl"`, the following are available:

`getGrouping signature(x = "ContControl")`: get slot `grouping`.

`setGrouping signature(x = "ContControl")`: set slot `grouping`.

`getAux signature(x = "ContControl")`: get slot `aux`.

`setAux signature(x = "ContControl")`: set slot `aux`.

Methods

In addition to the methods inherited from `"VirtualContControl"`, the following are available:

`contaminate signature(x = "data.frame", control = "ContControl")`: contaminate data.

`show signature(object = "ContControl")`: print the object on the R console.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

The slot `grouping` was named `group` prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named `getGroup` already exists.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`"DCARContControl"`, `"DARContControl"`, `"VirtualContControl"`, `contaminate`

Examples

```
showClass("ContControl")
```

DARContControl-class

Class "DARContControl"

Description

Class for controlling contamination in a simulation experiment. The values of the contaminated observations will be distributed at random (*DAR*), i.e., they will depend on on the original values.

Objects from the Class

Objects can be created by calls of the form `new("DARContControl", ...)`, `DARContControl(...)` or `ContControl(..., type="DAR")`.

Slots

`target`: Object of class `"OptCharacter"`; a character vector specifying specifying the variables (columns) to be contaminated, or `NULL` to contaminate all variables (except the additional ones generated internally).

`epsilon`: Object of class `"numeric"` giving the contamination levels.

`grouping`: Object of class `"character"` specifying a grouping variable (column) to be used for contaminating whole groups rather than individual observations.

`aux`: Object of class `"character"` specifying an auxiliary variable (column) whose values are used as probability weights for selecting the items (observations or groups) to be contaminated.

`fun`: Object of class `"function"` generating the values of the contamination data. The original values of the observations to be contaminated will be passed as its first argument. Furthermore, it should return an object that can be coerced to a `data.frame`, containing the contamination data.

`dots`: Object of class `"list"` containing additional arguments to be passed to `fun`.

Extends

Class `"ContControl"`, directly. Class `"VirtualContControl"`, by class "ContControl", distance 2. Class `"OptContControl"`, by class "ContControl", distance 3.

Details

With this control class, contamination is modeled as a two-step process. The first step is to select observations to be contaminated, the second is to model the distribution of the outliers. In this case, the original values will be modified by the function given by slot `fun`, i.e., values of the contaminated observations will depend on on the original values.

Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from `"ContControl"`, the following are available:

`getFun signature(x = "DARContControl")`: get slot `fun`.

`setFun signature(x = "DARContControl")`: set slot `fun`.

`getDots signature(x = "DARContControl")`: get slot `dots`.

`setDots signature(x = "DARContControl")`: set slot `dots`.

Methods

Methods are inherited from `"ContControl"`.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

The slot `grouping` was named `group` prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named `getGroup` already exists.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

Alfons, A., Templ, M. and Filzmoser, P. (2010) Contamination Models in the R Package **simFrame** for Statistical Simulation. In Aivazian, S., Filzmoser, P. and Kharin, Y. (editors) *Computer Data Analysis and Modeling: Complex Stochastic Data and Systems*, volume 2, 178–181. Minsk. ISBN 978-985-476-848-9.

Béguin, C. and Hulliger, B. (2008) The BACON-EEM Algorithm for Multivariate Outlier Detection in Incomplete Survey Data. *Survey Methodology*, **34**(1), 91–103.

Hulliger, B. and Schoch, T. (2009) Robust Multivariate Imputation with Survey Data. *57th Session of the International Statistical Institute*, Durban.

See Also

`"DCARContControl"`, `"ContControl"`, `"VirtualContControl"`, `contaminate`

Examples

```
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
foo <- generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
cc <- DARContControl(target = "V2",
    epsilon = 0.2, fun = function(x) x * 100)
contaminate(foo, cc)
```

DataControl-class *Class "DataControl"*

Description

Class for controlling model-based generation of data.

Objects from the Class

Objects can be created by calls of the form `new("DataControl", ...)` or `DataControl(...)`.

Slots

`size`: Object of class `"numeric"` giving the number of observations to be generated.

`distribution`: Object of class `"function"` generating the data, e.g., `rnorm` (the default) or `rmvnorm`. It should take a positive integer as its first argument, giving the number of observations to be generated, and return an object that can be coerced to a `data.frame`.

`dots`: Object of class `"list"` containing additional arguments to be passed to `distribution`.

`colnames`: Object of class `"OptCharacter"` ; a character vector to be used as column names for the generated `data.frame`, or `NULL`.

Extends

Class `"VirtualDataControl"`, directly. Class `"OptDataControl"`, by class "VirtualDataControl", distance 2.

Accessor and mutator methods

`getSize signature(x = "DataControl")`: get slot `size`.

`setSize signature(x = "DataControl")`: set slot `size`.

`getDistribution signature(x = "DataControl")`: get slot `distribution`.

`setDistribution signature(x = "DataControl")`: set slot `distribution`.

`getDots signature(x = "DataControl")`: get slot `dots`.

`setDots signature(x = "DataControl")`: set slot `dots`.

`getColnames signature(x = "DataControl")`: get slot `colnames`.

`setColnames signature(x = "DataControl")`: set slot `colnames`.

Methods

In addition to the methods inherited from `"VirtualDataControl"`, the following are available:

`generate signature(control = "DataControl")`: generate data.

`show signature(object = "DataControl")`: print the object on the R console.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL `http://www.jstatsoft.org/v37/i03/`.

See Also

"VirtualDataControl", generate

Examples

```
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
dc <- DataControl(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
generate(dc)
```

DCARContControl-class

Class "DCARContControl"

Description

Class for controlling contamination in a simulation experiment. The values of the contaminated observations will be distributed completely at random (*DCAR*), i.e., they will not depend on on the original values.

Objects from the Class

Objects can be created by calls of the form `new("DCARContControl", ...)`, `DCARContControl(...)` or `ContControl(..., type="DCAR")` (the latter exists mainly for back compatibility with early draft versions of `simFrame`).

Slots

`target`: Object of class `"OptCharacter"`; a character vector specifying specifying the variables (columns) to be contaminated, or `NULL` to contaminate all variables (except the additional ones generated internally).

`epsilon`: Object of class `"numeric"` giving the contamination levels.

`grouping`: Object of class `"character"` specifying a grouping variable (column) to be used for contaminating whole groups rather than individual observations (the same values are used for all observations in the same group).

`aux`: Object of class `"character"` specifying an auxiliary variable (column) whose values are used as probability weights for selecting the items (observations or groups) to be contaminated.

`distribution`: Object of class `"function"` generating the values of the contamination data, e.g., `rnorm` (the default) or `rmvnorm`. It should take a non-negative integer as its first argument, giving the number of items to be created, and return an object that can be coerced to a `data.frame`, containing the contamination data.

`dots`: Object of class `"list"` containing additional arguments to be passed to `distribution`.

Extends

Class "ContControl", directly. Class "VirtualContControl", by class "ContControl", distance 2. Class "OptContControl", by class "ContControl", distance 3.

Details

With this control class, contamination is modeled as a two-step process. The first step is to select observations to be contaminated, the second is to model the distribution of the outliers. In this case, the values of the contaminated observations will be generated by the function given by slot `fun` and will not depend on on the original values.

Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from "ContControl", the following are available:

`getDistribution signature(x = "DCARContControl")`: get slot `distribution`.

`setDistribution signature(x = "DCARContControl")`: set slot `distribution`.

`getDots signature(x = "DCARContControl")`: get slot `dots`.

`setDots signature(x = "DCARContControl")`: set slot `dots`.

Methods

Methods are inherited from "ContControl".

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

The slot `grouping` was named `group` prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named `getGroup` already exists.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

Alfons, A., Templ, M. and Filzmoser, P. (2010) Contamination Models in the R Package **simFrame** for Statistical Simulation. In Aivazian, S., Filzmoser, P. and Kharin, Y. (editors) *Computer Data*

Analysis and Modeling: Complex Stochastic Data and Systems, volume 2, 178–181. Minsk. ISBN 978-985-476-848-9.

Béguin, C. and Hulliger, B. (2008) The BACON-EEM Algorithm for Multivariate Outlier Detection in Incomplete Survey Data. *Survey Methodology*, **34**(1), 91–103.

Hulliger, B. and Schoch, T. (2009) Robust Multivariate Imputation with Survey Data. *57th Session of the International Statistical Institute*, Durban.

See Also

"DARContControl", "ContControl", "VirtualContControl", contaminate

Examples

```
data(eusilcP)
sam <- draw(eusilcP[, c("id", "eqIncome")], size = 20)
cc <- DCARContControl(target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000))
contaminate(sam, cc)
```

---

draw *Draw a sample*

Description

Generic function for drawing a sample.

Usage

```
draw(x, setup, ...)

## S4 method for signature 'data.frame,SampleSetup'
draw(x, setup, i = 1)

## S4 method for signature 'data.frame,VirtualSampleControl'
draw(x, setup)
```

Arguments

x	the data to sample from.
setup	an object of class "SampleSetup" containing previously set up samples, a control object inheriting from the virtual class "VirtualSampleControl" or a character string specifying such a control class (the default being "SampleControl").
i	an integer specifying which one of the previously set up samples should be drawn.
...	if setup is a character string or missing, the slots of the control object may be supplied as additional arguments. See "SampleControl" for details on the slots.

Value

A `data.frame` containing the sampled observations. In addition, the column `".weight"`, which consists of the sample weights, is added to the `data.frame`.

Methods

`x = "data.frame", setup = "character"` draw a sample using a control class specified by the character string `setup`. The slots of the control object may be supplied as additional arguments.

`x = "data.frame", setup = "missing"` draw a sample using a control object of class `"SampleControl"`. Its slots may be supplied as additional arguments.

`x = "data.frame", setup = "SampleSetup"` draw a previously set up sample.

`x = "data.frame", setup = "VirtualSampleControl"` draw a sample using a control object inheriting from the virtual class `"VirtualSampleControl"`.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`setup`, `"SampleSetup"`, `"SampleControl"`, `"TwoStageControl"`, `"VirtualSampleControl"`

Examples

```
## load data
data(eusilcP)

## simple random sampling
draw(eusilcP[, c("id", "eqIncome")], size = 20)

## group sampling
draw(eusilcP[, c("hid", "id", "eqIncome")],
    grouping = "hid", size = 10)

## stratified simple random sampling
draw(eusilcP[, c("id", "region", "eqIncome")],
    design = "region", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))

## stratified group sampling
draw(eusilcP[, c("hid", "id", "region", "eqIncome")],
    design = "region", grouping = "hid",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))
```

`eusilcP` *Synthetic EU-SILC data*

Description

This data set is synthetically generated from real Austrian EU-SILC (European Union Statistics on Income and Living Conditions) data.

Usage

```
data(eusilcP)
```

Format

A `data.frame` with 58 654 observations on the following 28 variables:

`hid` integer; the household ID.

`region` factor; the federal state in which the household is located (levels `Burgenland`, `Carinthia`, `Lower Austria`, `Salzburg`, `Styria`, `Tyrol`, `Upper Austria`, `Vienna` and `Vorarlberg`).

`hsize` integer; the number of persons in the household.

`eqsize` numeric; the equivalized household size according to the modified OECD scale.

`eqIncome` numeric; a simplified version of the equivalized household income.

`pid` integer; the personal ID.

id the household ID combined with the personal ID. The first five digits represent the household ID, the last two digits the personal ID (both with leading zeros).

`age` integer; the person's age.

`gender` factor; the person's gender (levels `male` and `female`).

`ecoStat` factor; the person's economic status (levels `1` = working full time, `2` = working part time, `3` = unemployed, `4` = pupil, student, further training or unpaid work experience or in compulsory military or community service, `5` = in retirement or early retirement or has given up business, `6` = permanently disabled or/and unfit to work or other inactive person, `7` = fulfilling domestic tasks and care responsibilities).

`citizenship` factor; the person's citizenship (levels `AT`, `EU` and `Other`).

`py010n` numeric; employee cash or near cash income (net).

`py050n` numeric; cash benefits or losses from self-employment (net).

`py090n` numeric; unemployment benefits (net).

`py100n` numeric; old-age benefits (net).

`py110n` numeric; survivor's benefits (net).

`py120n` numeric; sickness benefits (net).

`py130n` numeric; disability benefits (net).

`py140n` numeric; education-related allowances (net).

`hy040n` numeric; income from rental of a property or land (net).

`hy050n` numeric; family/children related allowances (net).

`hy070n` numeric; housing allowances (net).

`hy080n` numeric; regular inter-household cash transfer received (net).

`hy090n` numeric; interest, dividends, profit from capital investments in unincorporated business (net).

`hy110n` numeric; income received by people aged under 16 (net).

`hy130n` numeric; regular inter-household cash transfer paid (net).

`hy145n` numeric; repayments/receipts for tax adjustment (net).

`main` logical; indicates the main income holder (i.e., the person with the highest income) of each household.

Details

The data set is used as population data in some of the examples in package `simFrame`. Note that it is included for illustrative purposes only. It consists of 25 000 households, hence it does not represent the true population sizes of Austria and its regions.

Only a few of the large number of variables in the original survey are included in this example data set. Some variable names are different from the standardized names used by the statistical agencies, as the latter are rather cryptic codes. Furthermore, the variables `hsize`, `eqsize`, `eqIncome` and `age` are not included in the standardized format of EU-SILC data, but have been derived from other variables for convenience. Moreover, some very sparse income components were not included in the the generation of this synthetic data set. Thus the equivalized household income is computed from the available income components.

Source

This is a synthetic data set based on Austrian EU-SILC data from 2006. The original sample was provided by Statistics Austria.

References

Eurostat (2004) Description of target variables: Cross-sectional and longitudinal. *EU-SILC 065/04*, Eurostat.

Examples

```
data(eusilcP)
summary(eusilcP)

strata <- stratify(eusilcP, c("region", "gender"))
summary(strata)
```

`generate` *Generate data*

Description

Generic function for generating data based on a (distribution) model.

Usage

```
generate(control, ...)

## S4 method for signature 'DataControl'
generate(control)
```

Arguments

`control`	a control object inheriting from the virtual class `"VirtualDataControl"` or a character string specifying such a control class (the default being `"DataControl"`).
`...`	if `control` is a character string or missing, the slots of the control object may be supplied as additional arguments. See `"DataControl"` for details on the slots.

Details

The control class `"DataControl"` is quite simple but general. For user-defined data generation, it often suffices to implement a function and use it as the `distribution` slot in the `"DataControl"` object. See `"DataControl"` for some requirements for such a function.

However, if more specialized data generation models are required, the framework can be extended by defining a control class `"MyDataControl"` extending `"VirtualDataControl"` and the corresponding method `generate(control)` with signature `'MyDataControl'`. If, e.g., a specific distribution or mixture of distributions is frequently used in simulation experiments, a distinct control class may be more convenient for the user.

Value

A `data.frame`.

Methods

`control = "character"` generate data using a control class specified by the character string `control`. The slots of the control object may be supplied as additional arguments.

`control = "missing"` generate data using a control object of class `"DataControl"`. Its slots may be supplied as additional arguments.

`control = "DataControl"` generate data as defined by the control object `control`.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

"DataControl", "VirtualDataControl"

Examples

```
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)

# using a control object
dc <- DataControl(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
generate(dc)

# supply slots of control object as arguments
generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
```

---

head-methods *Methods for returning the first parts of an object*

Description

Return the first parts of an object.

Usage

```
## S4 method for signature 'SampleSetup'
head(x, k = 6, n = 6, ...)

## S4 method for signature 'SimControl'
head(x)

## S4 method for signature 'SimResults'
head(x, ...)

## S4 method for signature 'Strata'
head(x, ...)
```

```
## S4 method for signature 'VirtualContControl'
head(x)

## S4 method for signature 'VirtualDataControl'
head(x)

## S4 method for signature 'VirtualNAControl'
head(x)

## S4 method for signature 'VirtualSampleControl'
head(x)
```

Arguments

x	an object.
k	for objects of class "SampleSetup", the number of set up samples to be kept in the resulting object.
n	for objects of class "SampleSetup", the number of indices to be kept in each of the set up samples in the resulting object.
...	additional arguments to be passed down to methods.

Value

An object of the same class as x, but in general smaller. See the "Methods" section below for details.

Methods

signature(x = "SampleSetup") returns the first parts of set up samples. The first n indices of each of the first k set up samples are kept.

signature(x = "SimControl") currently returns the object itself.

signature(x = "SimResults") returns the first parts of simulation results. The method of head for the data.frame in slot values is thereby called.

signature(x = "Strata") returns the first parts of strata information. The method of head for the vector in slot values is thereby called and the slots split and size are adapted accordingly.

signature(x = "VirtualContControl") currently returns the object itself.

signature(x = "VirtualDataControl") currently returns the object itself.

signature(x = "VirtualNAControl") currently returns the object itself.

signature(x = "VirtualSampleControl") currently returns the object itself.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

head, "SampleSetup", "SimResults", "Strata"

Examples

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)
# get the first 10 indices of each of the first 5 samples
head(set, k = 5, n = 10)

## class "Strata"
# set up samples using group sampling
strata <- stratify(eusilcP, "region")
summary(strata)
# get strata information for the first 10 observations
head(strata, 10)
```

---

inclusionProb *Inclusion probabilities*

Description

Get the first-order inclusion probabilities from a vector of probability weights.

Usage

```
inclusionProb(prob, size)
```

Arguments

prob	a numeric vector of non-negative probability weights.
size	a non-negative integer giving the sample size.

Value

A numeric vector of the first-order inclusion probabilities.

Note

This is a faster C implementation of `inclusionprobabilities` from package `sampling`.

Author(s)

Andreas Alfons

See Also

`setup`, `"SampleSetup"`

Examples

```
pweights <- sample(1:5, 25, replace = TRUE)
inclusionProb(pweights, 10)
```

---

length-methods *Methods for getting the length of an object*

Description

Get the length of an object.

Usage

```
## S4 method for signature 'SampleSetup'
length(x)

## S4 method for signature 'VirtualContControl'
length(x)

## S4 method for signature 'VirtualNAControl'
length(x)

## S4 method for signature 'VirtualSampleControl'
length(x)
```

Arguments

`x`	an object.

Value

An integer giving the length of the object. See the “Methods” section below for details.

Methods

`signature(x = "SampleSetup")` get the number of set up samples.

`signature(x = "VirtualContControl")` get the number of contamination levels to be used.

`signature(x = "VirtualNAControl")` get the number of missing value rates to be used (the length in case of a vector in slot `NArate` or the number of rows in case of a matrix).

`signature(x = "VirtualSampleControl")` get the number of samples to be set up.

Author(s)

Andreas Alfons

See Also

`length`

Examples

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)
length(set)

## class "ContControl"
cc <- ContControl(target = "eqIncome",
    epsilon = c(0, 0.0025, 0.005, 0.0075, 0.01),
    dots = list(mean = 5e+05, sd = 10000))
length(cc)

## class "NAControl"
nc <- NAControl(target = "eqIncome", NArate = c(0.1, 0.2, 0.3))
length(nc)
```

NAControl-class *Class "NAControl"*

Description

Class for controlling the insertion of missing values in a simulation experiment.

Objects from the Class

Objects can be created by calls of the form `new("NAControl", ...)` or `NAControl(...)`.

Slots

`target`: Object of class `"OptCharacter"`; a character vector specifying the variables (columns) in which missing values should be inserted, or `NULL` to insert missing values in all variables (except the additional ones generated internally).

`NArate`: Object of class `"NumericMatrix"` giving the missing value rates, which may be selected individually for the target variables. In case of a vector, the same missing value rates are used for all target variables. In case of a matrix, on the other hand, the missing value rates to be used for each target variable are given by the respective column.

`grouping`: Object of class `"character"` specifying a grouping variable (column) to be used for setting whole groups to `NA` rather than individual values.

`aux`: Object of class `"character"` specifying auxiliary variables (columns) whose values are used as probability weights for selecting the values to be set to `NA` in the respective target variables. If only one variable (column) is specified, it is used for all target variables.

`intoContamination`: Object of class `"logical"` indicating whether missing values should also be inserted into contaminated observations. The default is to insert missing values only into non-contaminated observations.

Extends

Class `"VirtualNAControl"`, directly. Class `"OptNAControl"`, by class "VirtualNAControl", distance 2.

Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from `"VirtualNAControl"`, the following are available:

`getGrouping signature(x = "NAControl")`: get slot `grouping`.

`setGrouping signature(x = "NAControl")`: set slot `grouping`.

`getAux signature(x = "NAControl")`: get slot `aux`.

`setAux signature(x = "NAControl")`: set slot `aux`.

`getIntoContamination signature(x = "NAControl")`: get slot `intoContamination`.

`setIntoContamination signature(x = "NAControl")`: set slot `intoContamination`.

Methods

In addition to the methods inherited from `"VirtualNAControl"`, the following are available:

`setNA signature(x = "data.frame", control = "NAControl")`: set missing values.

`show signature(object = "NAControl")`: print the object on the R console.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

Since version 0.3, this control class now allows to specify an auxiliary variable with probability weights for each target variable.

The slot `grouping` was named `group` prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named `getGroup` already exists.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`"VirtualNAControl"`, `setNA`

Examples

```
data(eusilcP)
eusilcP$age[eusilcP$age < 0] <- 0  # this actually occurs
sam <- draw(eusilcP[, c("id", "age", "eqIncome")], size = 20)

## missing completely at random
mcarc <- NAControl(target = "eqIncome", NArate = 0.2)
setNA(sam, mcarc)

## missing at random
marc <- NAControl(target = "eqIncome", NArate = 0.2, aux = "age")
setNA(sam, marc)

## missing not at random
mnarc <- NAControl(target = "eqIncome",
    NArate = 0.2, aux = "eqIncome")
setNA(sam, mnarc)
```

NumericMatrix-class

Class "NumericMatrix"

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class `"NumericMatrix"` in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

Examples

```
showClass("NumericMatrix")
```

OptBasicVector-class

Class "OptBasicVector"

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class `"OptBasicVector"` in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

See Also

"SampleControl"

Examples

```
showClass("OptBasicVector")
```

`OptCall-class` *Class "OptCall"*

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "OptCall" in the signature.

Author(s)

Andreas Alfons

Examples

```
showClass("OptCall")
```

`OptCharacter-class` *Class "OptCharacter"*

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "OptCharacter" in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

Examples

```
showClass("OptCharacter")
```

OptContControl-class

Class "OptContControl"

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class `"OptContControl"` in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

See Also

`"SimControl"`

Examples

```
showClass("OptContControl")
```

`OptDataControl-class` *Class "OptDataControl"*

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class `"OptDataControl"` in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

See Also

`"SimResults"`

Examples

```
showClass("OptDataControl")
```

`OptNAControl-class` *Class "OptNAControl"*

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "OptNAControl" in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* simFrame. Use vignette("simFrame-intro") to view this vignette.

Author(s)

Andreas Alfons

See Also

"SimControl"

Examples

```
showClass("OptNAControl")
```

OptNumeric-class *Class "OptNumeric"*

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "OptNumeric" in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* simFrame. Use vignette("simFrame-intro") to view this vignette.

Author(s)

Andreas Alfons

Examples

```
showClass("OptNumeric")
```

`OptSampleControl-class` *Class "OptSampleControl"*

Description

Virtual class used internally for convenience.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class `"OptSampleControl"` in the signature.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

See Also

`"SimResults"`

Examples

```
showClass("OptSampleControl")
```

`plot-methods` *Plot simulation results*

Description

Plot simulation results. A suitable plot function is selected automatically, depending on the structure of the results.

Usage

```
## S4 method for signature 'SimResults,missing'
plot(x, y , ...)
```

Arguments

x	the simulation results.
y	not used.
...	further arguments to be passed to the selected plot function.

Value

An object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Details

The results of simulation experiments with at most one contamination level and at most one missing value rate are visualized by (conditional) box-and-whisker plots. For simulations involving different contamination levels or missing value rates, the average results are plotted against the contamination levels or missing value rates.

Methods

x = "SimResults", y = "missing" plot simulation results.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

simBwplot, simDensityplot, simXyplot, "SimResults"

Examples

```
#### design-based simulation
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome", epsilon = 0.02,
    fun = function(x) x * 25)

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
```

```
}

## run simulation
results <- runSimulation(eusilcP,
    sc, contControl = cc, fun = sim)

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)



#### model-based simulation
set.seed(12345)  # for reproducibility

## function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}

## run simulation
results <- runSimulation(dc, nrep = 50,
    contControl = cc, design = "group", fun = sim)

## plot results
plot(results, true = means)
```

`runSimulation` *Run a simulation experiment*

Description

Generic function for running a simulation experiment.

Usage

```
runSimulation(x, setup, nrep, control, contControl = NULL,
              NAControl = NULL, design = character(), fun, ...,
              SAE = FALSE)

runSim(...)
```

Arguments

x	a data.frame (for design-based simulation or simulation based on real data) or a control object for data generation inheriting from "VirtualDataControl" (for model-based simulation or mixed simulation designs).
setup	an object of class "SampleSetup", containing previously set up samples, or a control class for setting up samples inheriting from "VirtualSampleControl".
nrep	a non-negative integer giving the number of repetitions of the simulation experiment (for model-based simulation, mixed simulation designs or simulation based on real data).
control	a control object of class "SimControl"
contControl	an object of a class inheriting from "VirtualContControl", controlling contamination in the simulation experiment.
NAControl	an object of a class inheriting from "VirtualNAControl", controlling the insertion of missing values in the simulation experiment.
design	a character vector specifying variables (columns) to be used for splitting the data into domains. The simulations, including contamination and the insertion of missing values (unless SAE=TRUE), are then performed on every domain.
fun	a function to be applied in each simulation run.
...	for runSimulation, additional arguments to be passed to fun. For runSim, arguments to be passed to runSimulation.
SAE	a logical indicating whether small area estimation will be used in the simulation experiment.

Details

For convenience, the slots of control may be supplied as arguments.

There are some requirements for slot fun of the control object control. The function must return a numeric vector, or a list with the two components values (a numeric vector) and add (additional results of any class, e.g., statistical models). Note that the latter is computationally slightly more expensive. A data.frame is passed to fun in every simulation run. The corresponding argument must be called x. If comparisons with the original data need to be made, e.g., for evaluating the quality of imputation methods, the function should have an argument called orig. If different domains are used in the simulation, the indices of the current domain can be passed to the function via an argument called domain.

For small area estimation, the following points have to be kept in mind. The design for splitting the data must be supplied and SAE must be set to TRUE. However, the data are not actually split into the specified domains. Instead, the whole data set (sample) is passed to fun. Also contamination

and missing values are added to the whole data (sample). Last, but not least, the function must have a `domain` argument so that the current domain can be extracted from the whole data (sample).

In every simulation run, `fun` is evaluated using `try`. Hence no results are lost if computations fail in any of the simulation runs.

`runSim` is a wrapper for `runSimulation`.

Value

An object of class `"SimResults"`.

Methods

`x = "ANY", setup = "ANY", nrep = "ANY", control = "missing"` convenience wrapper that allows the slots of `control` to be supplied as arguments

`x = "data.frame", setup = "missing", nrep = "missing", control = "SimControl"` run a simulation experiment based on real data without repetitions (probably useless, but for completeness).

`x = "data.frame", setup = "missing", nrep = "numeric", control = "SimControl"` run a simulation experiment based on real data with repetitions.

`x = "data.frame", setup = "SampleSetup", nrep = "missing", control = "SimControl"` run a design-based simulation experiment with previously set up samples.

`x = "data.frame", setup = "VirtualSampleControl", nrep = "missing", control = "SimC` run a design-based simulation experiment.

`x = "VirtualDataControl", setup = "missing", nrep = "missing", control = "SimContrc` run a model-based simulation experiment without repetitions (probably useless, but for completeness).

`x = "VirtualDataControl", setup = "missing", nrep = "numeric", control = "SimContrc` run a model-based simulation experiment with repetitions.

`x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "missing", control` run a simulation experiment using a mixed simulation design without repetitions (probably useless, but for completeness).

`x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control` run a simulation experiment using a mixed simulation design with repetitions.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`"SimControl"`, `"SimResults"`, `simBwplot`, `simDensityplot`, `simXyplot`

Examples

```
#### design-based simulation
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome", epsilon = 0.02,
    fun = function(x) x * 25)

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
}

## run simulation and explore results
results <- runSimulation(eusilcP,
    sc, contControl = cc, fun = sim)
head(results)
aggregate(results)
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)



#### model-based simulation
set.seed(12345)  # for reproducibility

## function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}

## run simulation and explore results
results <- runSimulation(dc, nrep = 50,
    contControl = cc, design = "group", fun = sim)
head(results)
```

```
aggregate(results)
plot(results, true = means)
```

`SampleControl-class`
Class "SampleControl"

Description

Class for controlling the setup of samples.

Objects from the Class

Objects can be created by calls of the form `new("SampleControl", ...)` or `SampleControl(...)`.

Slots

`design`: Object of class `"BasicVector"` specifying variables (columns) to be used for stratified sampling.

`grouping`: Object of class `"BasicVector"` specifying a grouping variable (column) to be used for sampling whole groups rather than individual observations.

`collect`: Object of class `"logical"`; if a grouping variable is specified and this is `FALSE` (which is the default value), groups are sampled directly. If grouping variable is specified and this is `TRUE`, individuals are sampled in a first step. In a second step, all individuals that belong to the same group as any of the sampled individuals are collected and added to the sample. If no grouping variable is specified, this is ignored.

`fun`: Object of class `"function"` to be used for sampling (defaults to `srs`). It should return a vector containing the indices of the sampled items (observations or groups).

`size`: Object of class `"OptNumeric"`; an optional non-negative integer giving the number of items (observations or groups) to sample. In case of stratified sampling, a vector of non-negative integers, each giving the number of items to sample from the corresponding stratum, may be supplied.

`prob`: Object of class `"OptBasicVector"`; an optional numeric vector giving the probability weights, or a character string or logical vector specifying a variable (column) that contains the probability weights.

`dots`: Object of class `"list"` containing additional arguments to be passed to `fun`.

`k`: Object of class `"numeric"`; a single positive integer giving the number of samples to be set up.

Details

There are some restrictions on the argument names of the function supplied to `fun`. If it needs population data as input, the corresponding argument should be called `x` and should expect a `data.frame`. If the sampling method only needs the population size as input, the argument should be called `N`. Note that `fun` is not expected to have both `x` and `N` as arguments, and that the

latter is much faster for stratified sampling or group sampling. Furthermore, if the function has arguments for sample size and probability weights, they should be called `size` and `prob`, respectively. Note that a function with `prob` as its only argument is perfectly valid (for probability proportional to size sampling). Further arguments of `fun` may be supplied as a list via the slot `dots`.

Extends

Class "`VirtualSampleControl`", directly. Class "`OptSampleControl`", by class "VirtualSampleControl", distance 2.

Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from "`VirtualSampleControl`", the following are available:

`getDesign signature(x = "SampleControl")`: get slot `design`.
`setDesign signature(x = "SampleControl")`: set slot `design`.
`getGrouping signature(x = "SampleControl")`: get slot `grouping`.
`setGrouping signature(x = "SampleControl")`: set slot `grouping`.
`getCollect signature(x = "SampleControl")`: get slot `collect`.
`setCollect signature(x = "SampleControl")`: set slot `collect`.
`getFun signature(x = "SampleControl")`: get slot `fun`.
`setFun signature(x = "SampleControl")`: set slot `fun`.
`getSize signature(x = "SampleControl")`: get slot `size`.
`setSize signature(x = "SampleControl")`: set slot `size`.
`getProb signature(x = "SampleControl")`: get slot `prob`.
`setProb signature(x = "SampleControl")`: set slot `prob`.
`getDots signature(x = "SampleControl")`: get slot `dots`.
`setDots signature(x = "SampleControl")`: set slot `dots`.

Methods

In addition to the methods inherited from "`VirtualSampleControl`", the following are available:

`clusterSetup signature(cl = "ANY", x = "data.frame", control = "SampleControl")`: set up multiple samples on a `snow` cluster.
`setup signature(x = "data.frame", control = "SampleControl")`: set up multiple samples.
`show signature(object = "SampleControl")`: print the object on the R console.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

The slots `grouping` and `fun` were named `group` and `method`, respectively, prior to version 0.2. Renaming the slots was necessary since accessor and mutator functions were introduced in this version and functions named `getGroup`, `getMethod` and `setMethod` already exist.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`"VirtualSampleControl"`, `"TwoStageControl"`, `"SampleSetup"`, `setup`, `draw`

Examples

```
data(eusilcP)

## simple random sampling
srsc <- SampleControl(size = 20)
draw(eusilcP[, c("id", "eqIncome")], srsc)

## group sampling
gsc <- SampleControl(grouping = "hid", size = 10)
draw(eusilcP[, c("hid", "hid", "eqIncome")], gsc)

## stratified simple random sampling
ssrsc <- SampleControl(design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))
draw(eusilcP[, c("id", "region", "eqIncome")], ssrsc)

## stratified group sampling
sgsc <- SampleControl(design = "region", grouping = "hid",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgsc)
```

`SampleSetup-class` *Class "SampleSetup"*

Description

Class for set up samples.

Objects from the Class

Objects can be created by calls of the form `new("SampleSetup", ...)` or `SampleSetup(...)`.

However, objects are expected to be created by the function `setup` or `clusterSetup`, these constructor functions are not supposed to be called by the user.

Slots

`indices`: Object of class `"list"`; each list element contains the indices of the sampled observations.

`prob`: Object of class `"numeric"` giving the inclusion probabilities.

`control`: Object of class `"VirtualSampleControl"`; the control object used to set up the samples.

`seed`: Object of class `"list"` containing the seeds of the random number generator before and after setting up the samples, respectively (for replication purposes).

`call`: Object of class `"SimCall"`; the function call used to set up the samples, or `NULL`.

Accessor methods

`getIndices signature(x = "SampleSetup")`: get slot `indices`.

`getProb signature(x = "SampleSetup")`: get slot `prob`.

`getControl signature(x = "SampleSetup")`: get slot `control`.

`getSeed signature(x = "SampleSetup")`: get slot `seed`.

`getCall signature(x = "SampleSetup")`: get slot `call`.

Methods

`clusterRunSimulation signature(cl = "ANY", x = "data.frame", setup = "SampleSetup", nrep = "missing", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`draw signature(x = "data.frame", setup = "SampleSetup")`: draw a sample.

`head signature(x = "SampleSetup")`: returns the first parts of set up samples.

`length signature(x = "SampleSetup")`: get the number of set up samples.

`runSimulation signature(x = "data.frame", setup = "SampleSetup", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`show signature(object = "SampleSetup")`: print set up samples on the R console.

`summary signature(object = "SampleSetup")`: produce a summary of set up samples.

`tail signature(x = "SampleSetup")`: returns the last parts of set up samples.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

There are no mutator methods available since the slots are not supposed to be changed by the user.

Furthermore, the slot `seed` was added in version 0.2, and the slot `control` was added in version 0.3. Since the control object used to set up the samples is now stored, the redundant slots `design`, `grouping`, `collect` and `fun` were removed. This has been done as preparation for additional control classes for sampling, which will be introduced in future versions.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

"SampleControl", "TwoStageControl", "VirtualSampleControl", setup, draw

Examples

```
showClass("SampleSetup")
```

`sampling` *Random sampling*

Description

Functions for random sampling.

Usage

```
srs(N, size, replace = FALSE)

ups(N, size, prob, replace = FALSE)

brewer(prob, eps = 1e-06)

midzuno(prob, eps = 1e-06)

tille(prob, eps = 1e-06)
```

Arguments

`N`	a non-negative integer giving the number of observations from which to sample.
`size`	a non-negative integer giving the number of observations to sample.
`prob`	for `ups`, a numeric vector giving the probability weights (see `sample`). For `tille` and `midzuno`, a vector of inclusion probabilities (see `inclusionProb`).
`replace`	a logical indicating whether sampling should be performed with or without replacement.
`eps`	a numeric control value giving the desired accuracy.

Details

`srs` and `ups` are wrappers for simple random sampling and unequal probability sampling, respectively. Both functions make use of `sample`.

`brewer`, `midzuno` and `tille` perform Brewer's, Midzuno's and Tillé's method, respectively, for unequal probability sampling without replacement and fixed sample size.

Value

An integer vector giving the indices of the sampled observations.

Note

`brewer`, `midzuno` and `tille` are faster C implementations of `UPbrewer`, `UPmidzuno` and `UPtille`, respectively, from package `sampling`.

Author(s)

Andreas Alfons

References

Brewer, K. (1975), A simple procedure for sampling π pswor, Australian Journal of Statistics, **17**(3), 166-172.

Midzuno, H. (1952) On the sampling system with probability proportional to sum of size. *Annals of the Institute of Statistical Mathematics*, **3**(2), 99–107.

Tillé, Y. (1996) An elimination procedure of unequal probability sampling without replacement. *Biometrika*, **83**(1), 238–241.

Deville, J.-C. and Tillé, Y. (1998) Unequal probability sampling without replacement through a splitting method. *Biometrika*, **85**(1), 89–101.

See Also

`"SampleControl"`, `"TwoStageControl"`, `setup`, `inclusionProb`, `sample`, `UPbrewer`, `UPmidzuno`, `UPtille`

Examples

```
## simple random sampling
# without replacement
srs(10, 5)
# with replacement
srs(5, 10, replace = TRUE)

## unequal probability sampling
# without replacement
ups(10, 5, prob = 1:10)
# with replacement
ups(5, 10, prob = 1:5, replace = TRUE)

## Brewer, Midzuno and Tille sampling
# define inclusion probabilities
prob <- c(0.2,0.7,0.8,0.5,0.4,0.4)
# Brewer sampling
brewer(prob)
# Midzuno sampling
midzuno(prob)
# Tille sampling
tille(prob)
```

`setNA` *Set missing values*

Description

Generic function for inserting missing values into data.

Usage

```
setNA(x, control, ...)

## S4 method for signature 'data.frame,NAControl'
setNA(x, control, i)
```

Arguments

`x`	the data in which missing values should be inserted.
`control`	a control object inheriting from the virtual class `"VirtualNAControl"` or a character string specifying such a control class (the default being `"NAControl"`).
`i`	an integer giving the element or row of the slot `NArate` of `control` to be used as missing value rate(s).
`...`	if `control` is a character string or missing, the slots of the control object may be supplied as additional arguments. See `"NAControl"` for details on the slots.

Details

In order to extend the framework by a user-defined control class "MyNAControl" (which must extend "VirtualNAControl"), a method setNA(x, control, i) with signature 'data.frame, MyNAControl' needs to be implemented.

Value

A data.frame containing the data with missing values.

Methods

x = "data.frame", control = "character" set missing values using a control class specified by the character string control. The slots of the control object may be supplied as additional arguments.

x = "data.frame", control = "missing" set missing values using a control object of class "NAControl". Its slots may be supplied as additional arguments.

x = "data.frame", control = "NAControl" set missing values as defined by the control object control.

Note

Since version 0.3, setNA no longer checks if auxiliary variable(s) with probability weights are numeric and contain only finite positive values (sample still throws an error in these cases). This has been removed to improve computational performance in simulation studies.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

"NAControl", "VirtualNAControl"

Examples

```
data(eusilcP)
eusilcP$age[eusilcP$age < 0] <- 0  # this actually occurs
sam <- draw(eusilcP[, c("id", "age", "eqIncome")], size = 20)


## using control objects
# missing completely at random
mcarc <- NAControl(target = "eqIncome", NArate = 0.2)
setNA(sam, mcarc)
```

```
# missing at random
marc <- NAControl(target = "eqIncome", NArate = 0.2, aux = "age")
setNA(sam, marc)

# missing not at random
mnarc <- NAControl(target = "eqIncome",
    NArate = 0.2, aux = "eqIncome")
setNA(sam, mnarc)


## supply slots of control object as arguments
# missing completely at random
setNA(sam, target = "eqIncome", NArate = 0.2)

# missing at random
setNA(sam, target = "eqIncome", NArate = 0.2, aux = "age")

# missing not at random
setNA(sam, target = "eqIncome", NArate = 0.2, aux = "eqIncome")
```

`setup` *Set up multiple samples*

Description

Generic function for setting up multiple samples.

Usage

```
setup(x, control, ...)

## S4 method for signature 'data.frame,SampleControl'
setup(x, control)
```

Arguments

`x`	the data to sample from.
`control`	a control object inheriting from the virtual class `"VirtualSampleControl"` or a character string specifying such a control class (the default being `"SampleControl"`).
`...`	if `control` is a character string or missing, the slots of the control object may be supplied as additional arguments. See `"SampleControl"` for details on the slots.

Details

A fundamental design principle of the framework in the case of design-based simulation studies is that the sampling procedure is separated from the simulation procedure. Two main advantages arise from setting up all samples in advance.

First, the repeated sampling reduces overall computation time dramatically in certain situations, since computer-intensive tasks like stratification need to be performed only once. This is particularly relevant for large population data. In close-to-reality simulation studies carried out in research projects in survey statistics, often up to 10000 samples are drawn from a population of millions of individuals with stratified sampling designs. For such large data sets, stratification takes a considerable amount of time and is a very memory-intensive task. If the samples are taken on-the-fly, i.e., in every simulation run one sample is drawn, the function to take the stratified sample would typically split the population into the different strata in each of the 10000 simulation runs. If all samples are drawn in advance, on the other hand, the population data need to be split only once and all 10000 samples can be taken from the respective strata together.

Second, the samples can be stored permanently, which simplifies the reproduction of simulation results and may help to maximize comparability of results obtained by different partners in a research project. In particular, this is useful for large population data, when complex sampling techniques may be very time-consuming. In research projects involving different partners, usually different groups investigate different kinds of estimators. If the two groups use not only the same population data, but also the same previously set up samples, their results are highly comparable.

The control class `"SampleControl"` is highly flexible and allows stratified sampling as well as sampling of whole groups rather than individuals with a specified sampling method. Hence it is often sufficient to implement the desired sampling method for the simple non-stratified case to extend the existing framework. See `"SampleControl"` for some restrictions on the argument names of such a function, which should return a vector containing the indices of the sampled observations.

Nevertheless, for very complex sampling procedures, it is possible to define a control class `"MySampleControl"` extending `"VirtualSampleControl"`, and the corresponding method `setup(x, control)` with signature `'data.frame, MySampleControl'`. In order to optimize computational performance, it is necessary to efficiently set up multiple samples. Thereby the slot `k` of `"VirtualSampleControl"` needs to be used to control the number of samples, and the resulting object must be of class `"SampleSetup"`.

Value

An object of class `"SampleSetup"`.

Methods

`x = "data.frame", control = "character"` set up multiple samples using a control class specified by the character string `control`. The slots of the control object may be supplied as additional arguments.

`x = "data.frame", control = "missing"` set up multiple samples using a control object of class `"SampleControl"`. Its slots may be supplied as additional arguments.

`x = "data.frame", control = "SampleControl"` set up multiple samples as defined by the control object `control`.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

simSample, draw, "SampleControl", "TwoStageControl", "VirtualSampleControl", "SampleSetup"

Examples

```
data(eusilcP)

## simple random sampling
srss <- setup(eusilcP, size = 20, k = 4)
summary(srss)
draw(eusilcP[, c("id", "eqIncome")], srss, i = 1)

## group sampling
gss <- setup(eusilcP, grouping = "hid", size = 10, k = 4)
summary(gss)
draw(eusilcP[, c("hid", "id", "eqIncome")], gss, i = 2)

## stratified simple random sampling
ssrss <- setup(eusilcP, design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(ssrss)
draw(eusilcP[, c("id", "region", "eqIncome")], ssrss, i = 3)

## stratified group sampling
sgss <- setup(eusilcP, design = "region",
    grouping = "hid", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(sgss)
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgss, i = 4)
```

---

simApply *Apply a function to subsets*

Description

Generic functions for applying a function to subsets of a data set.

Usage

```
simApply(x, design, fun, ...)

simSapply(x, design, fun, ..., simplify = TRUE)
```

Arguments

x	the data.frame to be subsetted.
design	a character, logical or numeric vector specifying the variables (columns) used for subsetting.
fun	a function to be applied to the subsets.
simplify	a logical indicating whether the results should be simplified to a vector or matrix (if possible).
...	additional arguments to be passed to fun.

Value

For simApply a data.frame.

For simSapply, a list, vector or matrix (see sapply).

Methods for function simApply

x = "data.frame", design = "BasicVector", fun = "function" apply a function to subsets given by the variables (columns) in design.

x = "data.frame", design = "Strata", fun = "function" apply a function to subsets given by design.

Methods for function simSapply

x = "data.frame", design = "BasicVector", fun = "function" apply a function to subsets given by the variables (columns) in design.

x = "data.frame", design = "Strata", fun = "function" apply a function to subsets given by design.

Author(s)

Andreas Alfons

See Also

sapply

Examples

```
data(eusilcP)
eusilcP <- eusilcP[, c("region", "gender", "eqIncome")]

## returns data.frame
simApply(eusilcP, c("region", "gender"),
    function(x) median(x$eqIncome))

## returns vector
simSapply(eusilcP, c("region", "gender"),
    function(x) median(x$eqIncome))
```

`simBwplot` *Box-and-whisker plots*

Description

Generic function for producing box-and-whisker plots.

Usage

```
simBwplot(x, ...)

## S4 method for signature 'SimResults'
simBwplot(x, true = NULL, epsilon, NArate, select, ...)
```

Arguments

`x`	the object to be plotted. For plotting simulation results, this must be an object of class `"SimResults"`.
`true`	a numeric vector giving the true values. If supplied, reference lines are drawn in the corresponding panels.
`epsilon`	a numeric vector specifying contamination levels. If supplied, the values corresponding to these contamination levels are extracted from the simulation results and plotted.
`NArate`	a numeric vector specifying missing value rates. If supplied, the values corresponding to these missing value rates are extracted from the simulation results and plotted.
`select`	a character vector specifying the columns to be plotted. It must be a subset of the `colnames` slot of `x`, which is the default.
`...`	additional arguments to be passed down to methods and eventually to `bwplot`.

Details

For simulation results with multiple contamination levels or missing value rates, conditional box-and-whisker plots are produced.

Value

An object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Methods

x = "SimResults" produce box-and-whisker plots of simulation results.

Note

Functionality for producing conditional box-and-whisker plots was added in version 0.2. Prior to that, the function gave an error message if simulation results with multiple contamination levels or missing value rates were supplied.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

simDensityplot, simXyplot, bwplot, "SimResults"

Examples

```
#### design-based simulation
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome", epsilon = 0.02,
    fun = function(x) x * 25)

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
}

## run simulation
results <- runSimulation(eusilcP,
    sc, contControl = cc, fun = sim)

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
```

```
simBwplot(results, true = tv)



#### model-based simulation
set.seed(12345)  # for reproducibility

## function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}

## run simulation
results <- runSimulation(dc, nrep = 50,
    contControl = cc, design = "group", fun = sim)

## plot results
simBwplot(results, true = means)
```

`SimControl-class` *Class "SimControl"*

Description

Class for controlling how simulation runs are performed.

Objects from the Class

Objects can be created by calls of the form `new("SimControl", ...)` or `SimControl(...)`.

Slots

`contControl`**:** Object of class `"OptContControl"`; a control object for contamination, or `NULL`.

`NAControl`**:** Object of class `"OptNAControl"`; a control object for inserting missing values, or `NULL`.

`design`**:** Object of class `"character"` specifying variables (columns) to be used for splitting the data into domains. The simulations, including contamination and the insertion of missing values (unless `SAE=TRUE`), are then performed on every domain.

`fun`**:** Object of class `"function"` to be applied in each simulation run.

`dots`**:** Object of class `"list"` containing additional arguments to be passed to `fun`.

`SAE`**:** Object of class `"logical"` indicating whether small area estimation will be used in the simulation experiment.

Details

There are some requirements for `fun`. It must return a numeric vector, or a list with the two components `values` (a numeric vector) and `add` (additional results of any class, e.g., statistical models). Note that the latter is computationally slightly more expensive. A `data.frame` is passed to `fun` in every simulation run. The corresponding argument must be called `x`. If comparisons with the original data need to be made, e.g., for evaluating the quality of imputation methods, the function should have an argument called `orig`. If different domains are used in the simulation, the indices of the current domain can be passed to the function via an argument called `domain`.

For small area estimation, the following points have to be kept in mind. The `design` for splitting the data must be supplied and `SAE` must be set to `TRUE`. However, the data are not actually split into the specified domains. Instead, the whole data set (sample) is passed to `fun`. Also contamination and missing values are added to the whole data (sample). Last, but not least, the function must have a `domain` argument so that the current domain can be extracted from the whole data (sample).

In every simulation run, `fun` is evaluated using `try`. Hence no results are lost if computations fail in any of the simulation runs.

Accessor and mutator methods

`getContControl signature(x = "SimControl")`: get slot `ContControl`.

`setContControl signature(x = "SimControl")`: set slot `ContControl`.

`getNAControl signature(x = "SimControl")`: get slot `NAControl`.

`setNAControl signature(x = "SimControl")`: set slot `NAControl`.

`getDesign signature(x = "SimControl")`: get slot `design`.

`setDesign signature(x = "SimControl")`: set slot `design`.

`getFun signature(x = "SimControl")`: get slot `fun`.

`setFun signature(x = "SimControl")`: set slot `fun`.

`getDots signature(x = "SimControl")`: get slot `dots`.

`setDots signature(x = "SimControl")`: set slot `dots`.

`getSAE signature(x = "SimControl")`: get slot `SAE`.

`setSAE signature(x = "SimControl")`: set slot `SAE`.

SimControl-class 73

Methods

`clusterRunSimulation signature(cl = "ANY", x = "data.frame", setup = "missing", nrep = "numeric", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`clusterRunSimulation signature(cl = "ANY", x = "data.frame", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`clusterRunSimulation signature(cl = "ANY", x = "data.frame", setup = "SampleSetup", nrep = "missing", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`clusterRunSimulation signature(cl = "ANY", x = "VirtualDataControl", setup = "missing", nrep = "numeric", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`clusterRunSimulation signature(cl = "ANY", x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`head signature(x = "SimControl")`: currently returns the object itself.

`runSimulation signature(x = "data.frame", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "data.frame", setup = "SampleSetup", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "data.frame", setup = "missing", nrep = "numeric", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "data.frame", setup = "missing", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "missing", nrep = "numeric", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "missing", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`show signature(object = "SimControl")`: print the object on the R console.

`summary signature(object = "SimControl")`: currently returns the object itself.

`tail signature(x = "SimControl")`: currently returns the object itself.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

runSimulation, "SimResults"

Examples

```
#### design-based simulation
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome", epsilon = 0.02,
    fun = function(x) x * 25)

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
}

## combine these to "SimControl" object and run simulation
ctrl <- SimControl(contControl = cc, fun = sim)
results <- runSimulation(eusilcP, sc, control = ctrl)

## explore results
head(results)
aggregate(results)
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)



#### model-based simulation
set.seed(12345)  # for reproducibility

## function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

## control objects for data generation and contamination
means <- c(0, 0.25)
```

```
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}

## combine these to "SimControl" object and run simulation
ctrl <- SimControl(contControl = cc, design = "group", fun = sim)
results <- runSimulation(dc, nrep = 50, control = ctrl)

## explore results
head(results)
aggregate(results)
plot(results, true = means)
```

`simDensityplot` *Kernel density plots*

Description

Generic function for producing kernel density plots.

Usage

```
simDensityplot(x, ...)

## S4 method for signature 'SimResults'
simDensityplot(x, true = NULL, epsilon, NArate, select, ...)
```

Arguments

`x`	the object to be plotted. For plotting simulation results, this must be an object of class `"SimResults"`.
`true`	a numeric vector giving the true values. If supplied, reference lines are drawn in the corresponding panels.
`epsilon`	a numeric vector specifying contamination levels. If supplied, the values corresponding to these contamination levels are extracted from the simulation results and plotted.
`NArate`	a numeric vector specifying missing value rates. If supplied, the values corresponding to these missing value rates are extracted from the simulation results and plotted.

`select`	a character vector specifying the columns to be plotted. It must be a subset of the `colnames` slot of `x`, which is the default.
`...`	additional arguments to be passed down to methods and eventually to `densityplot`.

Details

For simulation results with multiple contamination levels or missing value rates, conditional kernel density plots are produced.

Value

An object of class `"trellis"`. The `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

Methods

`x = "SimResults"` produce kernel density plots of simulation results.

Note

Functionality for producing conditional kernel density plots was added in version 0.2. Prior to that, the function gave an error message if simulation results with multiple contamination levels or missing value rates were supplied.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`simBwplot`, `simXyplot`, `densityplot`, `"SimResults"`

Examples

```
#### design-based simulation
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome", epsilon = 0.02,
    fun = function(x) x * 25)

## function for simulation runs
```

```
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
}

## run simulation
results <- runSimulation(eusilcP,
    sc, contControl = cc, fun = sim)

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
simDensityplot(results, true = tv)



#### model-based simulation
set.seed(12345)  # for reproducibility

## function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}

## run simulation
results <- runSimulation(dc, nrep = 50,
    contControl = cc, design = "group", fun = sim)

## plot results
simDensityplot(results, true = means)
```

`SimResults-class` *Class "SimResults"*

Description

Class for simulation results.

Objects from the Class

Objects can be created by calls of the form `new("SimResults", ...)` or `SimResults(...)`.

However, objects are expected to be created by the function `runSimulation` or `clusterRunSimulation`, these constructor functions are not supposed to be called by the user.

Slots

`values`: Object of class `"data.frame"` containing the simulation results.

`add`: Object of class `"list"` containing additional simulation results, e.g., statistical models.

`design`: Object of class `"character"` giving the variables (columns) defining the domains used in the simulation experiment.

`colnames`: Object of class `"character"` giving the names of the columns of `values` that contain the actual simulation results.

`epsilon`: Object of class `"numeric"` containing the contamination levels used in the simulation experiment.

`NArate`: Object of class `"NumericMatrix"` containing the missing value rates used in the simulation experiment.

`dataControl`: Object of class `"OptDataControl"`; the control object used for data generation in model-based simulation, or `NULL`.

`sampleControl`: Object of class `"OptSampleControl"`; the control object used for sampling in design-based simulation, or `NULL`.

`nrep`: Object of class `"numeric"` giving the number of repetitions of the simulation experiment (for model-based simulation or simulation based on real data).

`control`: Object of class `"SimControl"`; the control object used for running the simulations.

`seed`: Object of class `"list"` containing the seeds of the random number generator before and after the simulation experiment, respectively (for replication of the results).

`call`: Object of class `"SimCall"`; the function call used to run the simulation experiment, or `NULL`.

Accessor methods

`getValues signature(x = "SimResults")`: get slot `values`.

`getAdd signature(x = "SimResults")`: get slot `add`.

`getDesign signature(x = "SimResults")`: get slot `design`.

`getColnames signature(x = "SimResults")`: get slot `colnames`.

`getEpsilon signature(x = "SimResults")`: get slot `epsilon`.

`getNArate signature(x = "SimResults")`: get slot `NArate`.

`getDataControl signature(x = "SimResults")`: get slot `dataControl`.

`getSampleControl signature(x = "SimResults")`: get slot `sampleControl`.

`getNrep signature(x = "SimResults")`: get slot `nrep`.

`getControl signature(x = "SimResults")`: get slot `control`.

`getSeed signature(x = "SimResults")`: get slot `seed`.

`getCall signature(x = "SimResults")`: get slot `call`.

Methods

`aggregate signature(x = "SimResults")`: aggregate simulation results.

`head signature(x = "SimResults")`: returns the first parts of simulation results.

`plot signature(x = "SimResults", y = "missing")`: selects a suitable graphical representation of the simulation results automatically.

`show signature(object = "SimResults")`: print simulation results on the R console.

`simBwplot signature(x = "SimResults")`: conditional box-and-whisker plot of simulation results.

`simDensityplot signature(x = "SimResults")`: conditional kernel density plot of simulation results.

`simXyplot signature(x = "SimResults")`: conditional x-y plot of simulation results.

`summary signature(x = "SimResults")`: produce a summary of simulation results.

`tail signature(x = "SimResults")`: returns the last parts of simulation results.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

There are no mutator methods available since the slots are not supposed to be changed by the user.

Furthermore, the slots `dataControl`, `sampleControl`, `nrep` and `control` were added in version 0.3.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`runSimulation`, `simBwplot`, `simDensityplot`, `simXyplot`

Examples

```
showClass("SimResults")
```

`simSample` *Set up multiple samples*

Description

A convenience wrapper for setting up multiple samples using `setup` with control class `SampleControl`.

Usage

```
simSample(x, design = character(), grouping = character(),
          collect = FALSE, fun = srs, size = NULL,
          prob = NULL, ..., k = 1)
```

Arguments

`x`	the `data.frame` to sample from.
`design`	a character, logical or numeric vector specifying variables (columns) to be used for stratified sampling.
`grouping`	a character string, single integer or logical vector specifying a grouping variable (column) to be used for sampling whole groups rather than individual observations.
`collect`	logical; if a grouping variable is specified and this is `FALSE` (which is the default value), groups are sampled directly. If grouping variable is specified and this is `TRUE`, individuals are sampled in a first step. In a second step, all individuals that belong to the same group as any of the sampled individuals are collected and added to the sample. If no grouping variable is specified, this is ignored.
`fun`	a function to be used for sampling (defaults to `srs`). It should return a vector containing the indices of the sampled items (observations or groups).
`size`	an optional non-negative integer giving the number of items (observations or groups) to sample. For stratified sampling, a vector of non-negative integers, each giving the number of items to sample from the corresponding stratum.
`prob`	an optional numeric vector giving the probability weights, or a character string or logical vector specifying a variable (column) that contains the probability weights.
`...`	additional arguments to be passed to `fun`.
`k`	a single positive integer giving the number of samples to be set up.

Details

There are some restrictions on the argument names of the function supplied to `fun`. If it needs population data as input, the corresponding argument should be called `x` and should expect a `data.frame`. If the sampling method only needs the population size as input, the argument should be called `N`. Note that `fun` is not expected to have both `x` and `N` as arguments, and that the latter is much faster for stratified sampling or group sampling. Furthermore, if the function has arguments for sample size and probability weights, they should be called `size` and `prob`, respectively.

Note that a function with `prob` as its only argument is perfectly valid (for probability proportional to size sampling). Further arguments of `fun` may be passed directly via the ... argument.

Value

An object of class `"SampleSetup"`.

Author(s)

Andreas Alfons

See Also

`setup`, `"SampleControl"`, `"SampleSetup"`

Examples

```
data(eusilcP)

## simple random sampling
srss <- simSample(eusilcP, size = 20, k = 4)
summary(srss)
draw(eusilcP[, c("id", "eqIncome")], srss, i = 1)

## group sampling
gss <- simSample(eusilcP, grouping = "hid", size = 10, k = 4)
summary(gss)
draw(eusilcP[, c("hid", "id", "eqIncome")], gss, i = 2)

## stratified simple random sampling
ssrss <- simSample(eusilcP, design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(ssrss)
draw(eusilcP[, c("id", "region", "eqIncome")], ssrss, i = 3)

## stratified group sampling
sgss <- simSample(eusilcP, design = "region",
    grouping = "hid", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(sgss)
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgss, i = 4)
```

`simXyplot` *X-Y plots*

Description

Generic function for producing x-y plots. For simulation results, the average results are plotted against the corresponding contamination levels or missing value rates.

Usage

```
simXyplot(x, ...)

## S4 method for signature 'SimResults'
simXyplot(x, true = NULL, epsilon, NArate,
          select, cond = c("Epsilon", "NArate"),
          average = c("mean", "median"), ...)
```

Arguments

x	the object to be plotted. For plotting simulation results, this must be an object of class "SimResults".
true	a numeric vector giving the true values. If supplied, reference lines are drawn in the corresponding panels.
epsilon	a numeric vector specifying contamination levels. If supplied, the values corresponding to these contamination levels are extracted from the simulation results and plotted.
NArate	a numeric vector specifying missing value rates. If supplied, the values corresponding to these missing value rates are extracted from the simulation results and plotted.
select	a character vector specifying the columns to be plotted. It must be a subset of the colnames slot of x, which is the default.
cond	a character string; for simulation results with multiple contamination levels and multiple missing value rates, this specifies the column of the simulation results to be used for producing conditional x-y plots. If "Epsilon", conditional plots are produced for the different contamination levels. If "NArate", conditional plots are produced for the different missing value rates. The default is to use whichever results in less plots.
average	a character string specifying how the averages should be computed. Possible values are "mean" for the mean (the default) or "median" for the median.
...	additional arguments to be passed down to methods and eventually to xyplot.

Details

For simulation results with multiple contamination levels and multiple missing value rates, conditional x-y plots are produced, as specified by cond.

Value

An object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

Methods

x = "SimResults" produce x-y plots of simulation results.

Note

Functionality for producing conditional x-y plots (including the argument `cond`) was added in version 0.2. Prior to that, the function gave an error message if simulation results with multiple contamination levels and multiple missing value rates were supplied.

The argument `average` that specifies how the averages are computed was added in version 0.1.2. Prior to that, the mean has always been used.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

simBwplot, simDensityplot, xyplot, "SimResults"

Examples

```
#### design-based simulation
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome",
    epsilon = seq(0, 0.05, by = 0.01),
    fun = function(x) x * 25)

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.05))
}

## run simulation
results <- runSimulation(eusilcP,
    sc, contControl = cc, fun = sim)

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
simXyplot(results, true = tv)



#### model-based simulation
set.seed(12345)  # for reproducibility
```

```
## function for generating data
rgnorm <- function(n, means) {
    group <- sample(1:2, n, replace=TRUE)
    data.frame(group=group, value=rnorm(n) + means[group])
}

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = seq(0, 0.05, by = 0.01),
    dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.05),
        median = median(x$value))
}

## run simulation
results <- runSimulation(dc, nrep = 50,
    contControl = cc, design = "group", fun = sim)

## plot results
simXyplot(results, true = means)
```

`Strata-class` *Class "Strata"*

Description

Class containing strata information for a data set.

Objects from the Class

Objects can be created by calls of the form `new("Strata", ...)` or `Strata(...)`.

However, objects are expected to be created by the function `stratify`, these constructor functions are not supposed to be called by the user.

Slots

`values`**:** Object of class `"integer"` giving the stratum number for each observation.

`split`**:** Object of class `"list"`; each list element contains the indices of the observations belonging to the corresponding stratum.

`design`**:** Object of class `"character"` giving the variables (columns) defining the strata.

`nr`**:** Object of class `"integer"` giving the stratum numbers.

`legend`**:** Object of class `"data.frame"` describing the strata.

`size`**:** Object of class `"numeric"` giving the stratum sizes.

`call`**:** Object of class `"OptCall"`; the function call used to stratify the data, or `NULL`.

Accessor methods

`getValues signature(x = "Strata")`: get slot `values`.

`getSplit signature(x = "Strata")`: get slot `split`.

`getDesign signature(x = "Strata")`: get slot `design`.

`getNr signature(x = "Strata")`: get slot `nr`.

`getLegend signature(x = "Strata")`: get slot `legend`.

`getSize signature(x = "Strata")`: get slot `size`.

`getCall signature(x = "Strata")`: get slot `call`.

Methods

`head signature(x = "Strata")`: returns the first parts of strata information.

`show signature(object = "Strata")`: print strata information on the R console.

`simApply signature(x = "data.frame", design = "Strata", fun = "function")`: apply a function to subsets.

`simSapply signature(x = "data.frame", design = "Strata", fun = "function")`: apply a function to subsets.

`summary signature(object = "Strata")`: produce a summary of strata information.

`tail signature(x = "Strata")`: returns the last parts of strata information.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

There are no mutator methods available since the slots are not supposed to be changed by the user.

Author(s)

Andreas Alfons

See Also

`stratify`

Examples

```
showClass("Strata")
```

`stratify` *Stratify data*

Description

Generic function for stratifying data.

Usage

```
stratify(x, design)
```

Arguments

`x`	the `data.frame` to be stratified.
`design`	a character, logical or numeric vector specifying the variables (columns) to be used for stratification.

Value

An object of class `"Strata"`.

Methods

`x = "data.frame", design = "BasicVector"` stratify data according to the variables (columns) given by `design`.

Author(s)

Andreas Alfons

See Also

`"Strata"`

Examples

```
data(eusilcP)
strata <- stratify(eusilcP, c("region", "gender"))
summary(strata)
```

`stratify-utilities` *Utility functions for stratifying data*

Description

Generic utility functions for stratifying data. These are useful if not all the information of class `"Strata"` is necessary.

Usage

```
getStrataLegend(x, design)

getStrataSplit(x, design, USE.NAMES = TRUE)

getStrataTable(x, design)

getStratumSizes(x, design, USE.NAMES = TRUE)

getStratumValues(x, design, split)
```

Arguments

`x`	the `data.frame` to be stratified. For `getStratumSizes`, it is also possible to supply a list in which each list element contains the indices of the observations belonging to the corresponding stratum (as returned by `getStrataSplit`).
`design`	a character, logical or numeric vector specifying the variables (columns) to be used for stratification.
`USE.NAMES`	a logical indicating whether information about the strata should be used as `names` for the result.
`split`	an optional list in which each list element contains the indices of the observations belonging to the corresponding stratum (as returned by `getStrataSplit`).

Value

For `getStrataLegend`, a `data.frame` describing the strata.

For `getStrataSplit`, a list in which each element contains the indices of the observations belonging to the corresponding stratum.

For `getStrataTable`, a `data.frame` describing the strata and containing the stratum sizes.

For `getStratumSizes`, a numeric vector of the stratum sizes.

For `getStratumValues`, a numeric vector giving the stratum number for each observation.

Methods for function getStrataLegend

x = "data.frame", design = "BasicVector" get a `data.frame` describing the strata, according to the variables specified by `design`.

Methods for function getStrataSplit

x = "data.frame", design = "BasicVector" get a list in which each element contains the indices of the observations belonging to the corresponding stratum, according to the variables specified by `design`.

Methods for function getStrataTable

x = "data.frame", design = "BasicVector" get a `data.frame` describing the strata and containing the stratum sizes, according to the variables specified by `design`.

Methods for function getStratumSizes

x = "list", design = "missing" get the stratum sizes for a list in which each list element contains the indices of the observations belonging to the corresponding stratum (as returned by `getStrataSplit`).

x = "data.frame", design = "BasicVector" get the stratum sizes of a data set, according to the variables specified by `design`.

Methods for function getStratumValues

x = "data.frame", design = "BasicVector", split = "list" get the stratum number for each observation, according to the variables specified by `design`. A previously computed list in which each list element contains the indices of the observations belonging to the corresponding stratum (as returned by `getStrataSplit`) speeds things up a bit.

x = "data.frame", design = "BasicVector", split = "missing" get the stratum number for each observation, according to the variables specified by `design`.

Author(s)

Andreas Alfons

See Also

`stratify`, `Strata`

Examples

```
data(eusilcP)

## all data
getStrataLegend(eusilcP, c("region", "gender"))
getStrataTable(eusilcP, c("region", "gender"))
getStratumSizes(eusilcP, c("region", "gender"))

## small sample
sam <- draw(eusilcP, size = 25)
getStrataSplit(sam, "gender")
getStratumValues(sam, "gender")
```

`summary-methods` *Methods for producing a summary of an object*

Description

Produce a summary an object.

Usage

```
## S4 method for signature 'SampleSetup'
summary(object)

## S4 method for signature 'SimControl'
summary(object)

## S4 method for signature 'SimResults'
summary(object, ...)

## S4 method for signature 'Strata'
summary(object)

## S4 method for signature 'VirtualContControl'
summary(object)

## S4 method for signature 'VirtualDataControl'
summary(object)

## S4 method for signature 'VirtualNAControl'
summary(object)

## S4 method for signature 'VirtualSampleControl'
summary(object)
```

Arguments

`object`	an object.
`...`	additional arguments to be passed down to methods.

Value

The form of the resulting object depends on the class of the argument `object`. See the "Methods" section below for details.

Methods

`signature(x = "SampleSetup")` returns an object of class `SummarySampleSetup`, which contains information on the size of each of the set up samples.

`signature(x = "SimControl")` currently returns the object itself.

`signature(x = "SimResults")` produces a summary of the simulation results by calling the method of `summary` for the `data.frame` in slot `values`.

`signature(x = "Strata")` returns a `data.frame` containing the size of each stratum.

`signature(x = "VirtualContControl")` currently returns the object itself.

`signature(x = "VirtualDataControl")` currently returns the object itself.

`signature(x = "VirtualNAControl")` currently returns the object itself.

`signature(x = "VirtualSampleControl")` currently returns the object itself.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`summary`, `"SampleSetup"`, `"SummarySampleSetup"`, `"SimResults"`, `"Strata"`

Examples

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)

## class "Strata"
# set up samples using group sampling
strata <- stratify(eusilcP, "region")
summary(strata)
```

SummarySampleSetup-class

Class "SummarySampleSetup"

Description

Class containing a summary of set up samples.

Objects from the Class

Objects can be created by calls of the form `new("SummarySampleSetup", ...)` or `SummarySampleSetup(...)`.

However, objects are expected to be created by the `summary` method for class `"SampleSetup"`, these constructor functions are not supposed to be called by the user.

Slots

`size`**:** Object of class `"numeric"` giving the size of each of the set up samples.

Accessor methods

`getSize signature(x = "SummarySampleSetup")`: get slot `size`.

Methods

`show signature(object = "SummarySampleSetup")`: print a summary of set up samples on the R console.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Note

There are no mutator methods available since the slots are not supposed to be changed by the user.

Author(s)

Andreas Alfons

See Also

`"SampleSetup"`, `summary`

Examples

```
showClass("SummarySampleSetup")
```

`tail-methods` *Methods for returning the last parts of an object*

Description

Return the last parts of an object.

Usage

```
## S4 method for signature 'SampleSetup'
tail(x, k = 6, n = 6, ...)

## S4 method for signature 'SimControl'
tail(x)

## S4 method for signature 'SimResults'
tail(x, ...)

## S4 method for signature 'Strata'
tail(x, ...)

## S4 method for signature 'VirtualContControl'
tail(x)

## S4 method for signature 'VirtualDataControl'
tail(x)

## S4 method for signature 'VirtualNAControl'
tail(x)

## S4 method for signature 'VirtualSampleControl'
tail(x)
```

Arguments

`x`	an object.
`k`	for objects of class `"SampleSetup"`, the number of set up samples to be kept in the resulting object.
`n`	for objects of class `"SampleSetup"`, the number of indices to be kept in each of the set up samples in the resulting object.
`...`	additional arguments to be passed down to methods.

Value

An object of the same class as `x`, but in general smaller. See the "Methods" section below for details.

Methods

`signature(x = "SampleSetup")` returns the last parts of set up samples. The last `n` indices of each of the last `k` set up samples are kept.

`signature(x = "SimControl")` currently returns the object itself.

`signature(x = "SimResults")` returns the last parts of simulation results. The method of `tail` for the `data.frame` in slot `values` is thereby called.

`signature(x = "Strata")` returns the last parts of strata information. The method of `tail` for the vector in slot `values` is thereby called and the slots `split` and `size` are adapted accordingly.

`signature(x = "VirtualContControl")` currently returns the object itself.

`signature(x = "VirtualDataControl")` currently returns the object itself.

`signature(x = "VirtualNAControl")` currently returns the object itself.

`signature(x = "VirtualSampleControl")` currently returns the object itself.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`tail`, `"SampleSetup"`, `"SimResults"`, `"Strata"`

Examples

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)
# get the last 10 indices of each of the last 5 samples
tail(set, k = 5, n = 10)

## class "Strata"
# set up samples using group sampling
strata <- stratify(eusilcP, "region")
summary(strata)
# get strata information for the last 10 observations
tail(strata, 10)
```

`TwoStageControl-class`
Class "TwoStageControl"

Description

Class for controlling the setup of samples using a two-stage procedure.

Usage

```
TwoStageControl(..., fun1 = srs, fun2 = srs, size1 = NULL,
                size2 = NULL, prob1 = NULL, prob2 = NULL,
                dots1 = list(), dots2 = list())
```

Arguments

`...`	the slots for the new object (see below).
`fun1`	the function to be used for sampling in the first stage (the first list component of slot `fun`).
`fun2`	the function to be used for sampling in the second stage (the second list component of slot `fun`).
`size1`	the number of PSUs to sample in the first stage (the first list component of slot `size`).
`size2`	the number of items to sample in the second stage (the second list component of slot `size`).
`prob1`	the probability weights for the first stage (the first list component of slot `prob`).
`prob2`	the probability weights for the second stage (the second list component of slot `prob`).
`dots1`	additional arguments to be passed to the function for sampling in the first stage (the first list component of slot `dots`).
`dots2`	additional arguments to be passed to the function for sampling in the second stage (the second list component of slot `dots`).

Objects from the Class

Objects can be created by calls of the form `new("TwoStageControl", ...)` or via the constructor `TwoStageControl`.

Slots

`design`**:** Object of class `"BasicVector"` specifying variables (columns) to be used for stratified sampling in the first stage.

`grouping`**:** Object of class `"BasicVector"` specifying grouping variables (columns) to be used for sampling primary sampling units (PSUs) and secondary sampling units (SSUs), respectively.

`fun`: Object of class `"list"`; a list of length two containing the functions to be used for sampling in the first and second stage, respectively (defaults to `srs` for both stages). The functions should return a vector containing the indices of the sampled items.

`size`: Object of class `"list"`; a list of length two, where each component contains an optional non-negative integer giving the number of items to sample in the first and second stage, respectively. In case of stratified sampling in the first stage, a vector of non-negative integers, each giving the number of PSUs to sample from the corresponding stratum, may be supplied. For the second stage, a vector of non-negative integers giving the number of items to sample from each PSU may be used.

`prob`: Object of class `"list"`; a list of length two, where each component gives optional probability weights for the first and second stage, respectively. Each component may thereby be a numerical vector, or a character string or integer vector specifying a variable (column) that contains the probability weights.

`dots`: Object of class `"list"`; a list of length two, where each component is again a list containing additional arguments to be passed to the corresponding function for sampling in `fun`.

`k`: Object of class `"numeric"`; a single positive integer giving the number of samples to be set up.

Details

There are some restrictions on the argument names of the functions for sampling in `fun`. If the sampling method needs population data as input, the corresponding argument should be called `x` and should expect a `data.frame`. If it only needs the population size as input, the argument should be called `N`. Note that the function is not expected to have both `x` and `N` as arguments, and that the latter is typically much faster. Furthermore, if the function has arguments for sample size and probability weights, they should be called `size` and `prob`, respectively. Note that a function with `prob` as its only argument is perfectly valid (for probability proportional to size sampling). Further arguments may be supplied as a list via the slot `dots`.

Extends

Class `"VirtualSampleControl"`, directly. Class `"OptSampleControl"`, by class "VirtualSampleControl", distance 2.

Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from `"VirtualSampleControl"`, the following are available:

`getDesign signature(x = "TwoStageControl")`: get slot `design`.

`setDesign signature(x = "TwoStageControl")`: set slot `design`.

`getGrouping signature(x = "TwoStageControl")`: get slot `grouping`.

`setGrouping signature(x = "TwoStageControl")`: set slot `grouping`.

`getCollect signature(x = "TwoStageControl")`: get slot `collect`.

`setCollect signature(x = "TwoStageControl")`: set slot `collect`.

`getFun signature(x = "TwoStageControl")`: get slot `fun`.

`setFun signature(x = "TwoStageControl")`: set slot `fun`.

`getSize signature(x = "TwoStageControl")`: get slot `size`.

`setSize signature(x = "TwoStageControl")`: set slot `size`.

`getProb signature(x = "TwoStageControl")`: get slot `prob`.

`setProb signature(x = "TwoStageControl")`: set slot `prob`.

`getDots signature(x = "TwoStageControl")`: get slot `dots`.

`setDots signature(x = "TwoStageControl")`: set slot `dots`.

Methods

In addition to the methods inherited from "VirtualSampleControl", the following are available:

`clusterSetup signature(cl = "ANY", x = "data.frame", control = "TwoStageControl")`: set up multiple samples on a `snow` cluster.

`setup signature(x = "data.frame", control = "TwoStageControl")`: set up multiple samples.

`show signature(object = "TwoStageControl")`: print the object on the R console.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

See Also

"VirtualSampleControl", "SampleControl", "SampleSetup", `setup`, `draw`

Examples

```
showClass("TwoStageControl")
```

`VirtualContControl-class` *Class "VirtualContControl"*

Description

Virtual superclass for controlling contamination in a simulation experiment.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`target`**:** Object of class `"OptCharacter"`; a character vector specifying specifying the variables (columns) to be contaminated, or `NULL` to contaminate all variables (except the additional ones generated internally).

`epsilon`**:** Object of class `"numeric"` giving the contamination levels.

Extends

Class `"OptContControl"`, directly.

Accessor and mutator methods

`getTarget signature(x = "VirtualContControl")`: get slot `target`.

`setTarget signature(x = "VirtualContControl")`: set slot `target`.

`getEpsilon signature(x = "VirtualContControl")`: get slot `epsilon`.

`setEpsilon signature(x = "VirtualContControl")`: set slot `epsilon`.

Methods

`head signature(x = "VirtualContControl")`: currently returns the object itself.

`length signature(x = "VirtualContControl")`: get the number of contamination levels to be used.

`show signature(object = "VirtualContControl")`: print the object on the R console.

`summary signature(object = "VirtualContControl")`: currently returns the object itself.

`tail signature(x = "VirtualContControl")`: currently returns the object itself.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

"DCARContControl", "DARContControl", "ContControl", contaminate

Examples

```
showClass("VirtualContControl")
```

VirtualDataControl-class

Class "VirtualDataControl"

Description

Virtual superclass for controlling model-based generation of data.

Objects from the Class

A virtual Class: No objects may be created from it.

Extends

Class "OptDataControl", directly.

Methods

clusterRunSimulation signature(cl = "ANY", x = "VirtualDataControl", setup = "missing", nrep = "numeric", control = "SimControl"): run a simulation experiment on a snow cluster.

clusterRunSimulation signature(cl = "ANY", x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control = "SimControl"): run a simulation experiment on a snow cluster.

head signature(x = "VirtualContControl"): currently returns the object itself.

runSimulation signature(x = "VirtualDataControl", setup = "missing", nrep = "numeric", control = "SimControl"): run a simulation experiment.

runSimulation signature(x = "VirtualDataControl", setup = "missing", nrep = "missing", control = "SimControl"): run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`summary signature(object = "VirtualContControl")`: currently returns the object itself.

`tail signature(x = "VirtualContControl")`: currently returns the object itself.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

"DataControl", generate

Examples

```
showClass("VirtualDataControl")
```

VirtualNAControl-class
Class "VirtualNAControl"

Description

Virtual superclass for controlling the insertion of missing values in a simulation experiment.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`target`: Object of class `"OptCharacter"`; a character vector specifying the variables (columns) in which missing values should be inserted, or `NULL` to insert missing values in all variables (except the additional ones generated internally).

`NArate`: Object of class `"NumericMatrix"` giving the missing value rates, which may be selected individually for the target variables. In case of a vector, the same missing value rates are used for all target variables. In case of a matrix, on the other hand, the missing value rates to be used for each target variable are given by the respective column.

Extends

Class `"OptNAControl"`, directly.

Accessor and mutator methods

`getTarget signature(x = "VirtualNAControl")`: get slot `target`.

`setTarget signature(x = "VirtualNAControl")`: set slot `target`.

`getNArate signature(x = "VirtualNAControl")`: get slot `NArate`.

`setNArate signature(x = "VirtualNAControl")`: set slot `NArate`.

Methods

`head signature(x = "VirtualNAControl")`: currently returns the object itself.

`length signature(x = "VirtualNAControl")`: get the number of missing value rates to be used (the length in case of a vector or the number of rows in case of a matrix).

`show signature(object = "VirtualNAControl")`: print the object on the R console.

`summary signature(object = "VirtualNAControl")`: currently returns the object itself.

`tail signature(x = "VirtualNAControl")`: currently returns the object itself.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

"NAControl", setNA

Examples

```
showClass("VirtualNAControl")
```

`VirtualSampleControl-class`
Class "VirtualSampleControl"

Description

Virtual superclass for controlling the setup of samples.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`k`: Object of class `"numeric"`, a single positive integer giving the number of samples to be set up.

Extends

Class `"OptSampleControl"`, directly.

Accessor and mutator methods

`getK signature(x = "VirtualSampleControl")`: get slot `k`.

`setK signature(x = "VirtualSampleControl")`: set slot `k`.

Methods

`clusterRunSimulation signature(cl = "ANY", x = "data.frame", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`clusterRunSimulation signature(cl = "ANY", x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control = "SimControl")`: run a simulation experiment on a `snow` cluster.

`draw signature(x = "data.frame", setup = "VirtualSampleControl")`: draw a sample.

`head signature(x = "VirtualSampleControl")`: currently returns the object itself.

`length signature(x = "VirtualSampleControl")`: get the number of samples to be set up.

`runSimulation signature(x = "data.frame", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control = "SimControl")`: run a simulation experiment.

`runSimulation signature(x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl")`: run a simulation experiment.

`show signature(object = "VirtualSampleControl")`: print the object on the R console.

`summary signature(object = "VirtualSampleControl")`: currently returns the object itself.

`tail signature(x = "VirtualSampleControl")`: currently returns the object itself.

UML class diagram

A slightly simplified UML class diagram of the framework can be found in Figure 1 of the package vignette *An Object-Oriented Framework for Statistical Simulation: The* R *Package* `simFrame`. Use `vignette("simFrame-intro")` to view this vignette.

Author(s)

Andreas Alfons

References

Alfons, A., Templ, M. and Filzmoser, P. (2010) An Object-Oriented Framework for Statistical Simulation: The R Package **simFrame**. *Journal of Statistical Software*, **37**(3), 1–36. URL http://www.jstatsoft.org/v37/i03/.

See Also

`"SampleControl"`, `"TwoStageControl"`, `"SampleSetup"`, `setup`, `draw`

Examples

```
showClass("VirtualSampleControl")
```

Index

∗Topic **attribute**
length-methods, 42
∗Topic **category**
aggregate-methods, 12
∗Topic **classes**
accessors, 5
BasicVector-class, 14
ContControl, 25
ContControl-class, 27
DARContControl-class, 28
DataControl-class, 30
DCARContControl-class, 32
NAControl-class, 43
NumericMatrix-class, 45
OptBasicVector-class, 46
OptCall-class, 47
OptCharacter-class, 47
OptContControl-class, 48
OptDataControl-class, 49
OptNAControl-class, 49
OptNumeric-class, 50
OptSampleControl-class, 51
SampleControl-class, 57
SampleSetup-class, 59
SimControl-class, 71
SimResults-class, 77
Strata-class, 84
SummarySampleSetup-class, 90
TwoStageControl-class, 94
VirtualContControl-class, 97
VirtualDataControl-class, 98
VirtualNAControl-class, 99
VirtualSampleControl-class, 101
∗Topic **datasets**
eusilcP, 36
∗Topic **design**
clusterRunSimulation, 16
runSimulation, 53
∗Topic **distribution**
clusterSetup, 20
draw, 34
generate, 38
inclusionProb, 41
sampling, 61
setup, 65
simSample, 80
∗Topic **hplot**
plot-methods, 51
simBwplot, 69
simDensityplot, 75
simXyplot, 81
∗Topic **iteration**
simApply, 67
∗Topic **manip**
contaminate, 23
head-methods, 39
setNA, 63
stratify, 86
stratify-utilities, 87
tail-methods, 92
∗Topic **methods**
accessors, 5
aggregate-methods, 12
clusterRunSimulation, 16
clusterSetup, 20
contaminate, 23
draw, 34
generate, 38
head-methods, 39
length-methods, 42
plot-methods, 51
runSimulation, 53
setNA, 63
setup, 65
simApply, 67
simBwplot, 69
simDensityplot, 75

simXyplot, 81
stratify, 86
stratify-utilities, 87
summary-methods, 89
tail-methods, 92
∗Topic **package**
simFrame-package, 3
∗Topic **survey**
inclusionProb, 41

accessors, 5
aggregate, *13*
aggregate,SimResults-method *(aggregate-methods)*, 12
aggregate-methods, 12
apply, *13*

BasicVector-class, 14
Basicvector-class *(BasicVector-class)*, 14
basicVector-class *(BasicVector-class)*, 14
basicvector-class *(BasicVector-class)*, 14
brewer *(sampling)*, 61
bwplot, *69, 70*

ClusterRunSimulation *(clusterRunSimulation)*, 16
ClusterRunsimulation *(clusterRunSimulation)*, 16
ClusterrunSimulation *(clusterRunSimulation)*, 16
Clusterrunsimulation *(clusterRunSimulation)*, 16
clusterRunSimulation, 16, *78*
clusterRunsimulation *(clusterRunSimulation)*, 16
clusterrunSimulation *(clusterRunSimulation)*, 16
clusterrunsimulation *(clusterRunSimulation)*, 16
clusterRunSimulation,ANY,ANY,ANY,ANY,missing-method *(clusterRunSimulation)*, 16
clusterRunSimulation,ANY,data.frame,missing,numeric,SimControl-method *(clusterRunSimulation)*, 16
clusterRunSimulation,ANY,data.frame,SampleControl,missing,SimControl-method *(clusterRunSimulation)*, 16
clusterRunSimulation,ANY,data.frame,VirtualSampl *(clusterRunSimulation)*, 16
clusterRunSimulation,ANY,VirtualDataControl,miss *(clusterRunSimulation)*, 16
clusterRunSimulation,ANY,VirtualDataControl,Virt *(clusterRunSimulation)*, 16
clusterRunSimulation-methods *(clusterRunSimulation)*, 16
ClusterSetup *(clusterSetup)*, 20
Clustersetup *(clusterSetup)*, 20
clusterSetup, 20, *60*
clustersetup *(clusterSetup)*, 20
clusterSetup,ANY,data.frame,character-method *(clusterSetup)*, 20
clusterSetup,ANY,data.frame,missing-method *(clusterSetup)*, 20
clusterSetup,ANY,data.frame,SampleControl-method *(clusterSetup)*, 20
clusterSetup,ANY,data.frame,TwoStageControl-meth *(clusterSetup)*, 20
clusterSetup-methods *(clusterSetup)*, 20
clusterSetupRNG, *18*, *22*
contaminate, 23, *28*, *30*, *34*, *98*
contaminate,data.frame,character-method *(contaminate)*, 23
contaminate,data.frame,ContControl-method *(contaminate)*, 23
contaminate,data.frame,missing-method *(contaminate)*, 23
contaminate-methods *(contaminate)*, 23
ContControl, *7, 8*, 25, *25, 26*, *29, 30*, *33, 34*, *98*
Contcontrol *(ContControl)*, 25
contControl *(ContControl)*, 25
contcontrol *(ContControl)*, 25
ContControl-class, 27
Contcontrol-class *(ContControl-class)*, 27
contControl-class *(ContControl-class)*, 27
contcontrol-class *(ContControl-class)*, 27
DARContControl, *7*, *23*, *25, 26*, *28*, *34*, *98*
DARContcontrol,missing,SimControl-method *(DARContControl-class)*, 28

DARContcontrol
(*DARContControl-class*), 28
DARcontControl
(*DARContControl-class*), 28
DARcontcontrol
(*DARContControl-class*), 28
darContcontrol
(*DARContControl-class*), 28
darcontControl
(*DARContControl-class*), 28
darcontcontrol
(*DARContControl-class*), 28
DARContControl-class, 28
DARContcontrol-class
(*DARContControl-class*), 28
DARcontControl-class
(*DARContControl-class*), 28
DARcontcontrol-class
(*DARContControl-class*), 28
darContcontrol-class
(*DARContControl-class*), 28
darcontControl-class
(*DARContControl-class*), 28
darcontcontrol-class
(*DARContControl-class*), 28
DataControl, *7*, *8*, *38*, *39*, *99*
DataControl (*DataControl-class*),
30
Datacontrol (*DataControl-class*),
30
dataControl (*DataControl-class*),
30
datacontrol (*DataControl-class*),
30
DataControl-class, 30
Datacontrol-class
(*DataControl-class*), 30
dataControl-class
(*DataControl-class*), 30
datacontrol-class
(*DataControl-class*), 30
DCARContControl, *7*, *23*, *25*, *26*, *28*, *30*, *98*
DCARContControl
(*DCARContControl-class*), 32
DCARContcontrol
(*DCARContControl-class*), 32
DCARcontControl
(*DCARContControl-class*), 32
DCARcontcontrol
(*DCARContControl-class*), 32
dcarContcontrol
(*DCARContControl-class*), 32
dcarcontControl
(*DCARContControl-class*), 32
dcarcontcontrol
(*DCARContControl-class*), 32
DCARContControl-class, 32
DCARContcontrol-class
(*DCARContControl-class*), 32
DCARcontControl-class
(*DCARContControl-class*), 32
DCARcontcontrol-class
(*DCARContControl-class*), 32
dcarContcontrol-class
(*DCARContControl-class*), 32
dcarcontControl-class
(*DCARContControl-class*), 32
dcarcontcontrol-class
(*DCARContControl-class*), 32
densityplot, *76*
draw, *22*, 34, *59*, *61*, *67*, *96*, *102*
draw,data.frame,character-method
(*draw*), 34
draw,data.frame,missing-method
(*draw*), 34
draw,data.frame,SampleSetup-method
(*draw*), 34
draw,data.frame,VirtualSampleControl-method
(*draw*), 34
draw-methods (*draw*), 34

eusilcP, 36
eusilcp (*eusilcP*), 36

generate, *32*, 38, *99*
generate,character-method
(*generate*), 38
generate,DataControl-method
(*generate*), 38
generate,missing-method
(*generate*), 38
generate-methods (*generate*), 38
getAdd (*accessors*), 5
getAdd,SimResults-method
(*SimResults-class*), 77
getAdd-methods (*accessors*), 5
getAux (*accessors*), 5

getAux,ContControl-method
(ContControl-class), 27
getAux,NAControl-method
(NAControl-class), 43
getAux-methods (accessors), 5
getCall (accessors), 5
getCall,SampleSetup-method
(SampleSetup-class), 59
getCall,SimResults-method
(SimResults-class), 77
getCall,Strata-method
(Strata-class), 84
getCall-methods (accessors), 5
getCollect (accessors), 5
getCollect,SampleControl-method
(SampleControl-class), 57
getCollect-methods (accessors), 5
getColnames (accessors), 5
getColnames,DataControl-method
(DataControl-class), 30
getColnames,SimResults-method
(SimResults-class), 77
getColnames-methods (accessors), 5
getContControl (accessors), 5
getContControl,SimControl-method
(SimControl-class), 71
getContControl-methods
(accessors), 5
getControl (accessors), 5
getControl,SampleSetup-method
(SampleSetup-class), 59
getControl,SimResults-method
(SimResults-class), 77
getControl-methods (accessors), 5
getDataControl (accessors), 5
getDataControl,SimResults-method
(SimResults-class), 77
getDataControl-methods
(accessors), 5
getDesign (accessors), 5
getDesign,SampleControl-method
(SampleControl-class), 57
getDesign,SimControl-method
(SimControl-class), 71
getDesign,SimResults-method
(SimResults-class), 77
getDesign,Strata-method
(Strata-class), 84
getDesign,TwoStageControl-method
(TwoStageControl-class), 94
getDesign-methods (accessors), 5
getDistribution (accessors), 5
getDistribution,DataControl-method
(DataControl-class), 30
getDistribution,DCARContControl-method
(DCARContControl-class), 32
getDistribution-methods
(accessors), 5
getDots (accessors), 5
getDots,DARContControl-method
(DARContControl-class), 28
getDots,DataControl-method
(DataControl-class), 30
getDots,DCARContControl-method
(DCARContControl-class), 32
getDots,SampleControl-method
(SampleControl-class), 57
getDots,SimControl-method
(SimControl-class), 71
getDots,TwoStageControl-method
(accessors), 5
getDots-methods (accessors), 5
getEpsilon (accessors), 5
getEpsilon,SimResults-method
(SimResults-class), 77
getEpsilon,VirtualContControl-method
(VirtualContControl-class),
97
getEpsilon-methods (accessors), 5
getFun (accessors), 5
getFun,DARContControl-method
(DARContControl-class), 28
getFun,SampleControl-method
(SampleControl-class), 57
getFun,SimControl-method
(SimControl-class), 71
getFun,TwoStageControl-method
(accessors), 5
getFun-methods (accessors), 5
getGrouping (accessors), 5
getGrouping,ContControl-method
(ContControl-class), 27
getGrouping,NAControl-method
(NAControl-class), 43
getGrouping,SampleControl-method
(SampleControl-class), 57

getGrouping,TwoStageControl-method
(TwoStageControl-class), 94
getGrouping-methods (accessors), 5
getIndices (accessors), 5
getIndices,SampleSetup-method
(SampleSetup-class), 59
getIndices-methods (accessors), 5
getIntoContamination (accessors),
5
getIntoContamination,NAControl-method
(NAControl-class), 43
getIntoContamination-methods
(accessors), 5
getK (accessors), 5
getK,VirtualSampleControl-method
(VirtualSampleControl-class),
101
getK-methods (accessors), 5
getLegend (accessors), 5
getLegend,Strata-method
(Strata-class), 84
getLegend-methods (accessors), 5
getNAControl (accessors), 5
getNAControl,SimControl-method
(SimControl-class), 71
getNAControl-methods (accessors),
5
getNArate (accessors), 5
getNArate,SimResults-method
(SimResults-class), 77
getNArate,VirtualNAControl-method
(VirtualNAControl-class),
99
getNArate-methods (accessors), 5
getNr (accessors), 5
getNr,Strata-method
(Strata-class), 84
getNr-methods (accessors), 5
getNrep (accessors), 5
getNrep,SimResults-method
(SimResults-class), 77
getNrep-methods (accessors), 5
getProb (accessors), 5
getProb,SampleControl-method
(SampleControl-class), 57
getProb,SampleSetup-method
(SampleSetup-class), 59
getProb,TwoStageControl-method
(accessors), 5
getProb-methods (accessors), 5
getSAE (accessors), 5
getSAE,SimControl-method
(SimControl-class), 71
getSAE-methods (accessors), 5
getSampleControl (accessors), 5
getSampleControl,SimResults-method
(SimResults-class), 77
getSampleControl-methods
(accessors), 5
getSeed (accessors), 5
getSeed,SampleSetup-method
(SampleSetup-class), 59
getSeed,SimResults-method
(SimResults-class), 77
getSeed-methods (accessors), 5
getSize (accessors), 5
getSize,DataControl-method
(DataControl-class), 30
getSize,SampleControl-method
(SampleControl-class), 57
getSize,Strata-method
(Strata-class), 84
getSize,SummarySampleSetup-method
(SummarySampleSetup-class),
90
getSize,TwoStageControl-method
(accessors), 5
getSize-methods (accessors), 5
getSplit (accessors), 5
getSplit,Strata-method
(Strata-class), 84
getSplit-methods (accessors), 5
getStrataLegend
(stratify-utilities), 87
getStrataLegend,data.frame,BasicVector-method
(stratify-utilities), 87
getStrataLegend-methods
(stratify-utilities), 87
getStrataSplit
(stratify-utilities), 87
getStrataSplit,data.frame,BasicVector-method
(stratify-utilities), 87
getStrataSplit-methods
(stratify-utilities), 87
getStrataTable
(stratify-utilities), 87

getStrataTable,data.frame,BasicVector-method
(stratify-utilities), 87
getStrataTable-methods
(stratify-utilities), 87
getStratumSizes
(stratify-utilities), 87
getStratumSizes,data.frame,BasicVector-method
(stratify-utilities), 87
getStratumSizes,list,missing-method
(stratify-utilities), 87
getStratumSizes-methods
(stratify-utilities), 87
getStratumValues
(stratify-utilities), 87
getStratumValues,data.frame,BasicVector,list-method
(stratify-utilities), 87
getStratumValues,data.frame,BasicVector,missing-method
(stratify-utilities), 87
getStratumValues-methods
(stratify-utilities), 87
getTarget (accessors), 5
getTarget,VirtualContControl-method
(VirtualContControl-class),
97
getTarget,VirtualNAControl-method
(VirtualNAControl-class),
99
getTarget-methods (accessors), 5
getValues (accessors), 5
getValues,SimResults-method
(SimResults-class), 77
getValues,Strata-method
(Strata-class), 84
getValues-methods (accessors), 5

head, 40, 41
head,SampleSetup-method
(head-methods), 39
head,SimControl-method
(head-methods), 39
head,SimResults-method
(head-methods), 39
head,Strata-method
(head-methods), 39
head,VirtualContControl-method
(head-methods), 39
head,VirtualDataControl-method
(head-methods), 39
head,VirtualNAControl-method
(head-methods), 39
head,VirtualSampleControl-method
(head-methods), 39
head-methods, 39

InclusionProb (inclusionProb), 41
inclusionProb, 41, 62
inclusionprob (inclusionProb), 41
inclusionprobabilities, 42

length, 43
length,SampleSetup-method
(length-methods), 42
length,VirtualContControl-method
(length-methods), 42
length,VirtualNAControl-method
(length-methods), 42
length,VirtualSampleControl-method
(length-methods), 42
length-methods, 42

makeCluster, 18, 22
midzuno (sampling), 61

NAControl, 7, 8, 63, 64, 101
NAControl (NAControl-class), 43
NAcontrol (NAControl-class), 43
naControl (NAControl-class), 43
nacontrol (NAControl-class), 43
NAControl-class, 43
NAcontrol-class
(NAControl-class), 43
naControl-class
(NAControl-class), 43
nacontrol-class
(NAControl-class), 43
NumericMatrix-class, 45
Numericmatrix-class
(NumericMatrix-class), 45
numericMatrix-class
(NumericMatrix-class), 45
numericmatrix-class
(NumericMatrix-class), 45

OptBasicVector, 15
OptBasicVector-class, 46
OptBasicvector-class
(OptBasicVector-class), 46

OptbasicVector-class
(OptBasicVector-class), 46
optBasicVector-class
(OptBasicVector-class), 46
optBasicvector-class
(OptBasicVector-class), 46
optbasicVector-class
(OptBasicVector-class), 46
optbasicvector-class
(OptBasicVector-class), 46
OptCall-class, 47
Optcall-class (OptCall-class), 47
optCall-class (OptCall-class), 47
optcall-class (OptCall-class), 47
OptCharacter-class, 47
Optcharacter-class
(OptCharacter-class), 47
optCharacter-class
(OptCharacter-class), 47
optcharacter-class
(OptCharacter-class), 47
OptContControl, *27*, *29*, *33*, *97*
OptContControl-class, 48
OptContcontrol-class
(OptContControl-class), 48
OptcontControl-class
(OptContControl-class), 48
Optcontcontrol-class
(OptContControl-class), 48
optContControl-class
(OptContControl-class), 48
optContcontrol-class
(OptContControl-class), 48
optcontControl-class
(OptContControl-class), 48
optcontcontrol-class
(OptContControl-class), 48
OptDataControl, *31*, *98*
OptDataControl-class, 49
OptDatacontrol-class
(OptDataControl-class), 49
OptdataControl-class
(OptDataControl-class), 49
Optdatacontrol-class
(OptDataControl-class), 49
optDataControl-class
(OptDataControl-class), 49
optDatacontrol-class
(OptDataControl-class), 49
optdataControl-class
(OptDataControl-class), 49
optdatacontrol-class
(OptDataControl-class), 49
OptNAControl, *44*, *100*
OptNAControl-class, 49
OptNAcontrol-class
(OptNAControl-class), 49
OptnaControl-class
(OptNAControl-class), 49
Optnacontrol-class
(OptNAControl-class), 49
optNAControl-class
(OptNAControl-class), 49
optNAcontrol-class
(OptNAControl-class), 49
optnaControl-class
(OptNAControl-class), 49
optnacontrol-class
(OptNAControl-class), 49
OptNumeric-class, 50
Optnumeric-class
(OptNumeric-class), 50
optNumeric-class
(OptNumeric-class), 50
optnumeric-class
(OptNumeric-class), 50
OptSampleControl, *58*, *95*, *101*
OptSampleControl-class, 51
OptSamplecontrol-class
(OptSampleControl-class), 51
OptsampleControl-class
(OptSampleControl-class), 51
Optsamplecontrol-class
(OptSampleControl-class), 51
optSampleControl-class
(OptSampleControl-class), 51
optSamplecontrol-class
(OptSampleControl-class), 51
optsampleControl-class
(OptSampleControl-class), 51

optsamplecontrol-class
(OptSampleControl-class), 51
Optsbasicvector-class
(OptBasicVector-class), 46

plot,SimResults,missing-method
(plot-methods), 51
plot-methods, 51
print, 52, 70, 76, 82

rmvnorm, 31, 32
rnorm, 31, 32
runSim (runSimulation), 53
RunSimulation (runSimulation), 53
Runsimulation (runSimulation), 53
runSimulation, 18, 53, 74, 78, 79
runsimulation (runSimulation), 53
runSimulation,ANY,ANY,ANY,missing-method
(runSimulation), 53
runSimulation,data.frame,missing,missing,SimControl-method
(runSimulation), 53
runSimulation,data.frame,missing,numeric,SimControl-method
(runSimulation), 53
runSimulation,data.frame,SampleSetup,missing,SimControl-method
(runSimulation), 53
runSimulation,data.frame,VirtualSampleControl,missing,SimControl-method
(runSimulation), 53
runSimulation,VirtualDataControl,missing,missing,SimControl-method
(runSimulation), 53
runSimulation,VirtualDataControl,missing,numeric,SimControl-method
(runSimulation), 53
runSimulation,VirtualDataControl,VirtualSampleControl,missing,SimControl-method
(runSimulation), 53
runSimulation,VirtualDataControl,VirtualSampleControl,numeric,SimControl-method
(runSimulation), 53
RunSimulation-methods
(runSimulation), 53
Runsimulation-methods
(runSimulation), 53
runSimulation-methods
(runSimulation), 53
runsimulation-methods
(runSimulation), 53

sample, 24, 62, 64
SampleControl, 7, 8, 20–22, 34, 35, 47, 61, 62, 65–67, 80, 81, 96, 102
SampleControl
(SampleControl-class), 57
Samplecontrol
(SampleControl-class), 57
sampleControl
(SampleControl-class), 57
samplecontrol
(SampleControl-class), 57
SampleControl-class, 57
Samplecontrol-class
(SampleControl-class), 57
sampleControl-class
(SampleControl-class), 57
samplecontrol-class
(SampleControl-class), 57
SampleSetup, 21, 22, 35, 41, 42, 59, 66, 67, 81, 90, 91, 93, 96, 102
SampleSetup (SampleSetup-class), 59
Samplesetup (SampleSetup-class), 59
sampleSetup (SampleSetup-class), 59
samplesetup (SampleSetup-class), 59
SampleSetup-class, 59
Samplesetup-class
(SampleSetup-class), 59
sampleSetup-class
(SampleSetup-class), 59
samplesetup-class
(SampleSetup-class), 59
sampling, 61
sapply, 68
setAux (accessors), 5
setAux,ContControl-method
(ContControl-class), 27
setAux,NAControl-method
(NAControl-class), 43
setAux-methods (accessors), 5
setCollect (accessors), 5
setCollect,SampleControl-method
(SampleControl-class), 57
setCollect-methods (accessors), 5
setColnames (accessors), 5
setColnames,DataControl-method
(DataControl-class), 30
setColnames-methods (accessors), 5

setContControl (accessors), 5
setContControl,SimControl-method (SimControl-class), 71
setContControl-methods (accessors), 5
setDesign (accessors), 5
setDesign,SampleControl-method (SampleControl-class), 57
setDesign,SimControl-method (SimControl-class), 71
setDesign,TwoStageControl-method (TwoStageControl-class), 94
setDesign-methods (accessors), 5
setDistribution (accessors), 5
setDistribution,DataControl-method (DataControl-class), 30
setDistribution,DCARContControl-method (DCARContControl-class), 32
setDistribution-methods (accessors), 5
setDots (accessors), 5
setDots,DARContControl-method (DARContControl-class), 28
setDots,DataControl-method (DataControl-class), 30
setDots,DCARContControl-method (DCARContControl-class), 32
setDots,SampleControl-method (SampleControl-class), 57
setDots,SimControl-method (SimControl-class), 71
setDots,TwoStageControl-method (accessors), 5
setDots-methods (accessors), 5
setEpsilon (accessors), 5
setEpsilon,VirtualContControl-method (VirtualContControl-class), 97
setEpsilon-methods (accessors), 5
setFun (accessors), 5
setFun,DARContControl-method (DARContControl-class), 28
setFun,SampleControl-method (SampleControl-class), 57
setFun,SimControl-method (SimControl-class), 71
setFun,TwoStageControl-method (accessors), 5
setFun-methods (accessors), 5
setGrouping (accessors), 5
setGrouping,ContControl-method (ContControl-class), 27
setGrouping,NAControl-method (NAControl-class), 43
setGrouping,SampleControl-method (SampleControl-class), 57
setGrouping,TwoStageControl-method (TwoStageControl-class), 94
setGrouping-methods (accessors), 5
setIntoContamination (accessors), 5
setIntoContamination,NAControl-method (NAControl-class), 43
setIntoContamination-methods (accessors), 5
setK (accessors), 5
setK,VirtualSampleControl-method (VirtualSampleControl-class), 101
setK-methods (accessors), 5
SetNA (setNA), 63
Setna (setNA), 63
setNA, *45*, 63, *101*
setna (setNA), 63
setNA,data.frame,character-method (setNA), 63
setNA,data.frame,missing-method (setNA), 63
setNA,data.frame,NAControl-method (setNA), 63
SetNA-methods (setNA), 63
Setna-methods (setNA), 63
setNA-methods (setNA), 63
setna-methods (setNA), 63
setNAControl (accessors), 5
setNAControl,SimControl-method (SimControl-class), 71
setNAControl-methods (accessors), 5
setNArate (accessors), 5
setNArate,VirtualNAControl-method (VirtualNAControl-class), 99
setNArate-methods (accessors), 5
setProb (accessors), 5
setProb,SampleControl-method

(SampleControl-class), 57
setProb,TwoStageControl-method (accessors), 5
setProb-methods (accessors), 5
setSAE (accessors), 5
setSAE,SimControl-method (SimControl-class), 71
setSAE-methods (accessors), 5
setSize (accessors), 5
setSize,DataControl-method (DataControl-class), 30
setSize,SampleControl-method (SampleControl-class), 57
setSize,TwoStageControl-method (accessors), 5
setSize-methods (accessors), 5
setTarget (accessors), 5
setTarget,VirtualContControl-method (VirtualContControl-class), 97
setTarget,VirtualNAControl-method (VirtualNAControl-class), 99
setTarget-methods (accessors), 5
setup, 22, 35, 42, 59–62, 65, 80, 81, 96, 102
setup,data.frame,character-method (setup), 65
setup,data.frame,missing-method (setup), 65
setup,data.frame,SampleControl-method (setup), 65
setup,data.frame,TwoStageControl-method (setup), 65
setup-methods (setup), 65
show,ContControl-method (ContControl-class), 27
show,DataControl-method (DataControl-class), 30
show,NAControl-method (NAControl-class), 43
show,SampleControl-method (SampleControl-class), 57
show,SampleSetup-method (SampleSetup-class), 59
show,SimControl-method (SimControl-class), 71
show,SimResults-method (SimResults-class), 77
show,Strata-method (Strata-class), 84
show,SummarySampleSetup-method (SummarySampleSetup-class), 90
show,TwoStageControl-method (TwoStageControl-class), 94
show,VirtualContControl-method (VirtualContControl-class), 97
show,VirtualNAControl-method (VirtualNAControl-class), 99
show,VirtualSampleControl-method (VirtualSampleControl-class), 101
SimApply (simApply), 67
Simapply (simApply), 67
simApply, 67
simapply (simApply), 67
simApply,data.frame,BasicVector,function-method (simApply), 67
simApply,data.frame,Strata,function-method (simApply), 67
SimApply-methods (simApply), 67
Simapply-methods (simApply), 67
simApply-methods (simApply), 67
simapply-methods (simApply), 67
SimBwplot (simBwplot), 69
simBwplot, 18, 52, 55, 69, 76, 79, 83
simbwplot (simBwplot), 69
simBwplot,SimResults-method (simBwplot), 69
SimBwplot-methods (simBwplot), 69
simBwplot-methods (simBwplot), 69
simbwplot-methods (simBwplot), 69
SimControl, 7, 8, 18, 48, 50, 55
SimControl (SimControl-class), 71
Simcontrol (SimControl-class), 71
simControl (SimControl-class), 71
simcontrol (SimControl-class), 71
SimControl-class, 71
Simcontrol-class (SimControl-class), 71
simControl-class (SimControl-class), 71
simcontrol-class (SimControl-class), 71

SimDensityplot (simDensityplot), 75
Simdensityplot (simDensityplot), 75
simDensityplot, 18, 52, 55, 70, 75, 79, 83
simdensityplot (simDensityplot), 75
simDensityplot,SimResults-method (simDensityplot), 75
SimDensityplot-methods (simDensityplot), 75
Simdensityplot-methods (simDensityplot), 75
simDensityplot-methods (simDensityplot), 75
simdensityplot-methods (simDensityplot), 75
SimFrame (simFrame-package), 3
Simframe (simFrame-package), 3
simFrame (simFrame-package), 3
simframe (simFrame-package), 3
SimFrame-package (simFrame-package), 3
Simframe-package (simFrame-package), 3
simFrame-package, 3
simframe-package (simFrame-package), 3
SimResults, 13, 18, 41, 49, 51, 52, 55, 70, 74, 76, 83, 90, 93
SimResults (SimResults-class), 77
Simresults (SimResults-class), 77
simResults (SimResults-class), 77
simresults (SimResults-class), 77
SimResults-class, 77
Simresults-class (SimResults-class), 77
simResults-class (SimResults-class), 77
simresults-class (SimResults-class), 77
SimSample (simSample), 80
Simsample (simSample), 80
simSample, 67, 80
simsample (simSample), 80
SimSapply (simApply), 67
Simsapply (simApply), 67
simSapply (simApply), 67
simsapply (simApply), 67
simSapply,data.frame,BasicVector,function-method (simApply), 67
simSapply,data.frame,Strata,function-method (simApply), 67
SimSapply-methods (simApply), 67
Simsapply-methods (simApply), 67
simSapply-methods (simApply), 67
simsapply-methods (simApply), 67
SimXyplot (simXyplot), 81
Simxyplot (simXyplot), 81
simXyplot, 18, 52, 55, 70, 76, 79, 81
simxyplot (simXyplot), 81
simXyplot,SimResults-method (simXyplot), 81
SimXyplot-methods (simXyplot), 81
Simxyplot-methods (simXyplot), 81
simXyplot-methods (simXyplot), 81
simxyplot-methods (simXyplot), 81
srs, 57, 80, 95
srs (sampling), 61
Strata, 41, 86, 88, 90, 93
Strata (Strata-class), 84
strata (Strata-class), 84
Strata-class, 84
strata-class (Strata-class), 84
stratify, 84, 85, 86, 88
stratify,data.frame,BasicVector-method (stratify), 86
stratify-methods (stratify), 86
stratify-utilities, 87
summary, 90, 91
summary,SampleSetup-method (summary-methods), 89
summary,SimControl-method (summary-methods), 89
summary,SimResults-method (summary-methods), 89
summary,Strata-method (summary-methods), 89
summary,VirtualContControl-method (summary-methods), 89
summary,VirtualDataControl-method (summary-methods), 89
summary,VirtualNAControl-method (summary-methods), 89
summary,VirtualSampleControl-method (summary-methods), 89

summary-methods, 89
SummarySampleSetup, *90*
SummarySampleSetup (*SummarySampleSetup-class*), 90
SummarySamplesetup (*SummarySampleSetup-class*), 90
SummarysampleSetup (*SummarySampleSetup-class*), 90
Summarysamplesetup (*SummarySampleSetup-class*), 90
summarySampleSetup (*SummarySampleSetup-class*), 90
summarySamplesetup (*SummarySampleSetup-class*), 90
summarysampleSetup (*SummarySampleSetup-class*), 90
summarysamplesetup (*SummarySampleSetup-class*), 90
SummarySampleSetup-class, 90
SummarySamplesetup-class (*SummarySampleSetup-class*), 90
SummarysampleSetup-class (*SummarySampleSetup-class*), 90
Summarysamplesetup-class (*SummarySampleSetup-class*), 90
summarySampleSetup-class (*SummarySampleSetup-class*), 90
summarySamplesetup-class (*SummarySampleSetup-class*), 90
summarysampleSetup-class (*SummarySampleSetup-class*), 90
summarysamplesetup-class (*SummarySampleSetup-class*), 90
tail, *93*
tail,SampleSetup-method (*tail-methods*), 92
tail,SimControl-method (*tail-methods*), 92
tail,SimResults-method (*tail-methods*), 92
tail,Strata-method (*tail-methods*), 92
tail,VirtualContControl-method (*tail-methods*), 92
tail,VirtualDataControl-method (*tail-methods*), 92
tail,VirtualNAControl-method (*tail-methods*), 92
tail,VirtualSampleControl-method (*tail-methods*), 92
tail-methods, 92
tille (*sampling*), 61
TwoStageControl, *7*, *8*, *22*, *35*, *59*, *61*, *62*, *67*, *102*
TwoStageControl (*TwoStageControl-class*), 94
TwoStagecontrol (*TwoStageControl-class*), 94
TwostageControl (*TwoStageControl-class*), 94
Twostagecontrol (*TwoStageControl-class*), 94
twoStageControl (*TwoStageControl-class*), 94
twoStagecontrol (*TwoStageControl-class*), 94
twostageControl (*TwoStageControl-class*), 94
twostagecontrol (*TwoStageControl-class*), 94
TwoStageControl-class, 94
TwoStagecontrol-class (*TwoStageControl-class*), 94
TwostageControl-class (*TwoStageControl-class*), 94
Twostagecontrol-class (*TwoStageControl-class*), 94
twoStageControl-class (*TwoStageControl-class*), 94
twoStagecontrol-class (*TwoStageControl-class*), 94

twostageControl-class
(TwoStageControl-class), 94
twostagecontrol-class
(TwoStageControl-class), 94

UPbrewer, *62*
update, *52*, *70*, *76*, *82*
UPmidzuno, *62*
ups *(sampling)*, 61
UPtille, *62*

VirtualContControl, *7*, *8*, *24*, *25*, *27–30*, *33*, *34*
VirtualContControl-class, 97
VirtualContcontrol-class
(VirtualContControl-class), 97
VirtualcontControl-class
(VirtualContControl-class), 97
Virtualcontcontrol-class
(VirtualContControl-class), 97
virtualContControl-class
(VirtualContControl-class), 97
virtualContcontrol-class
(VirtualContControl-class), 97
virtualcontControl-class
(VirtualContControl-class), 97
virtualcontcontrol-class
(VirtualContControl-class), 97
VirtualDataControl, *31*, *32*, *38*, *39*
VirtualDataControl-class, 98
VirtualDatacontrol-class
(VirtualDataControl-class), 98
VirtualdataControl-class
(VirtualDataControl-class), 98
Virtualdatacontrol-class
(VirtualDataControl-class), 98
virtualDataControl-class
(VirtualDataControl-class), 98
virtualDatacontrol-class
(VirtualDataControl-class), 98
virtualdataControl-class
(VirtualDataControl-class), 98
virtualdatacontrol-class
(VirtualDataControl-class), 98
VirtualNAControl, *8*, *44*, *45*, *64*
VirtualNAControl-class, 99
VirtualNAcontrol-class
(VirtualNAControl-class), 99
VirtualnaControl-class
(VirtualNAControl-class), 99
Virtualnacontrol-class
(VirtualNAControl-class), 99
virtualNAControl-class
(VirtualNAControl-class), 99
virtualNAcontrol-class
(VirtualNAControl-class), 99
virtualnaControl-class
(VirtualNAControl-class), 99
virtualnacontrol-class
(VirtualNAControl-class), 99
VirtualSampleControl, *8*, *21*, *22*, *35*, *58*, *59*, *61*, *66*, *67*, *95*, *96*
VirtualSampleControl-class, 101
VirtualSamplecontrol-class
(VirtualSampleControl-class), 101
VirtualsampleControl-class
(VirtualSampleControl-class), 101
Virtualsamplecontrol-class
(VirtualSampleControl-class), 101
virtualSampleControl-class
(VirtualSampleControl-class), 101
virtualSamplecontrol-class

(*VirtualSampleControl-class*), 101
virtualsampleControl-class (*VirtualSampleControl-class*), 101
virtualsamplecontrol-class (*VirtualSampleControl-class*), 101

xyplot, *82, 83*

Package 'simPopulation'

October 19, 2010

Type Package

Title Simulation of synthetic populations for surveys based on sample data

Version 0.2.1

Date 2010-10-19

Author Andreas Alfons and Stefan Kraft

Maintainer Andreas Alfons <alfons@statistik.tuwien.ac.at>

Depends R(>= 2.7.1), nnet, POT, lattice, vcd

Imports lattice, vcd

Description Simulate populations for surveys based on sample data with special application to EU-SILC.

License GPL (>= 2)

LazyLoad yes

Repository CRAN

Date/Publication 2010-10-19 10:00:39

R topics documented:

simPopulation-package 2
contingencyWt 3
eusilcS 5
getBreaks 6
getCat 8
quantileWt 9
simCategorical 10
simComponents 11
simContinuous 13
simEUSILC 17

simStructure 20
spBwplot 21
spBwplotStats 23
spCdf 24
spCdfplot 25
spMosaic 27
spTable 28
tableWt 30
utils 31

Index **33**

`simPopulation-package`

Simulation of synthetic populations for surveys based on sample data

---

Description

Simulate populations for surveys based on sample data with special application to EU-SILC.

Details

Package:	simPopulation
Type:	Package
Version:	0.2.1
Date:	2010-10-19
Depends:	R(>= 2.7.1), nnet, POT, lattice, vcd
Imports:	lattice, vcd
License:	GPL (>= 2)
LazyLoad:	yes

Index:

`contingencyWt` Weighted contingency coefficients
`eusilcS` Synthetic EU-SILC survey data
`getBreaks` Compute breakpoints for categorizing (semi-)continuous variables
`getCat` Categorize (semi-)continuous variables
`meanWt` Weighted mean, variance, covariance matrix and correlation matrix
`quantileWt` Weighted sample quantiles
`simCategorical` Simulate categorical variables of population data
`simComponents` Simulate components of continuous variables of population data
`simContinuous` Simulate continuous variables of population

	data
simEUSILC	Simulate EU-SILC population data
simPopulation-package	Simulation of synthetic populations for surveys based on sample data
simStructure	Simulate the household structure of population data
spBwplot	Weighted box plots
spBwplotStats	Weighted box plot statistics
spCdf	(Weighted empirical) cumulative distribution function
spCdfplot	Plot (weighted empirical) cumulative distribution functions
spMosaic	Mosaic plots of expected and realized population sizes
spTable	Cross tabulations of expected and realized population sizes
tableWt	Weighted cross tabulation

Further information is available in the following vignettes:

simPopulation-eusilc	Simulation of EU-SILC Population Data: Using the R Package simPopulation (source, pdf)

Author(s)

Andreas Alfons and Stefan Kraft

Maintainer: Andreas Alfons <alfons@statistik.tuwien.ac.at>

`contingencyWt` *Weighted contingency coefficients*

Description

Compute (weighted) pairwise contingency coefficients.

Usage

```
contingencyWt(x, ...)

## Default S3 method:
contingencyWt(x, y, weights = NULL, ...)

## S3 method for class 'matrix':
contingencyWt(x, weights = NULL, ...)

## S3 method for class 'data.frame':
contingencyWt(x, weights = NULL, ...)
```

Arguments

x	for the default method, a vector that can be interpreted as factor. For the matrix and data.frame methods, the columns should be interpretable as factors.
y	a vector that can be interpreted as factor.
weights	an optional numeric vector containing sample weights.
...	for the generic function, arguments to be passed down to the methods, otherwise ignored.

Details

The function tableWt is used for the computation of the corresponding pairwise contingency tables.

Value

For the default method, the (weighted) contingency coefficient of x and y.

For the matrix and data.frame method, a matrix of (weighted) pairwise contingency coefficients for all combinations of columns. Elements below the diagonal are NA.

Author(s)

Andreas Alfons and Stefan Kraft

References

Kendall, M.G. and Stuart, A. (1967) *The Advanced Theory of Statistics, Volume 2: Inference and Relationship*. Charles Griffin & Co Ltd, London, 2nd edition.

See Also

tableWt

Examples

```
data(eusilcS)

## default method
contingencyWt(eusilcS$pl030, eusilcS$pb220a, weights = eusilcS$rb050)

## data.frame method
basic <- c("age", "rb090", "hsize", "pl030", "pb220a")
contingencyWt(eusilcS[, basic], weights = eusilcS$rb050)
```

`eusilcS` *Synthetic EU-SILC survey data*

Description

This data set is synthetically generated from real Austrian EU-SILC (European Union Statistics on Income and Living Conditions) data.

Usage

```
data(eusilcS)
```

Format

A data frame with 11725 observations on the following 18 variables.

`db030` integer; the household ID.

`hsize` integer; the number of persons in the household.

`db040` factor; the federal state in which the household is located (levels `Burgenland`, `Carinthia`, `Lower Austria`, `Salzburg`, `Styria`, `Tyrol`, `Upper Austria`, `Vienna` and `Vorarlberg`).

`age` integer; the person's age.

`rb090` factor; the person's gender (levels `male` and `female`).

`pl030` factor; the person's economic status (levels `1` = working full time, `2` = working part time, `3` = unemployed, `4` = pupil, student, further training or unpaid work experience or in compulsory military or community service, `5` = in retirement or early retirement or has given up business, `6` = permanently disabled or/and unfit to work or other inactive person, `7` = fulfilling domestic tasks and care responsibilities).

`pb220a` factor; the person's citizenship (levels `AT`, `EU` and `Other`).

`netIncome` numeric; the personal net income.

`py010n` numeric; employee cash or near cash income (net).

`py050n` numeric; cash benefits or losses from self-employment (net).

`py090n` numeric; unemployment benefits (net).

`py100n` numeric; old-age benefits (net).

`py110n` numeric; survivor's benefits (net).

`py120n` numeric; sickness benefits (net).

`py130n` numeric; disability benefits (net).

`py140n` numeric; education-related allowances (net).

`db090` numeric; the household sample weights.

`rb050` numeric; the personal sample weights.

Details

The data set consists of 4641 households and is used as sample data in some of the examples in package `simPopulation`. Note that it is included for illustrative purposes only. The sample weights do not reflect the true population sizes of Austria and its regions. The resulting population data is about 100 times smaller than the real population size to save computation time.

Only a few of the large number of variables in the original survey are included in this example data set. The variable names are rather cryptic codes, but these are the standardized names used by the statistical agencies. Furthermore, the variables `hsize`, `age` and `netIncome` are not included in the standardized format of EU-SILC data, but have been derived from other variables for convenience.

Source

This is a synthetic data set based on Austrian EU-SILC data from 2006. The original sample was provided by Statistics Austria.

References

Eurostat (2004) Description of target variables: Cross-sectional and longitudinal. *EU-SILC 065/04*, Eurostat.

Examples

```
data(eusilcS)
summary(eusilcS)
```

`getBreaks` *Compute breakpoints for categorizing (semi-)continuous variables*

Description

Compute breakpoints for categorizing continuous or semi-continuous variables using (weighted) quantiles. This is a utility function that is useful for writing custom wrapper functions such as `simEUSILC`.

Usage

```
getBreaks(x, weights = NULL, zeros = TRUE,
          lower = NULL, upper = NULL, equidist = TRUE)
```

Arguments

`x`	a numeric vector to be categorized.
`weights`	an optional numeric vector containing sample weights.
`zeros`	a logical indicating whether `x` is semi-continuous, i.e., contains a considerable amount of zeros. See "Details" on how this affects the behavior of the function.

`lower, upper` optional numeric values specifying lower and upper bounds other than minimum and maximum of `x`, respectively.

`equidist` a logical indicating whether the (positive) breakpoints should be equidistant or whether there should be refinements in the lower and upper tail (see "Details").

Details

If `equidist` is `TRUE`, the behavior is as follows. If `zeros` is `TRUE` as well, the 0%, 10%, ..., 90% quantiles of the negative values and the 10%, 20%, ..., 100% of the positive values are computed. These quantiles are then used as breakpoints together with 0. If `zeros` is not `TRUE`, on the other hand, the 0%, 10%, ..., 100% quantiles of all values are used.

If `equidist` is not `TRUE`, the behavior is as follows. If `zeros` is not `TRUE`, the 1%, 5%, 10%, 20%, 40%, 60%, 80%, 90%, 95% and 99% quantiles of all values are used for the inner part of the data (instead of the equidistant 10%, ..., 90% quantiles). If `zeros` is `TRUE`, these quantiles are only used for the positive values while the quantiles of the negative values remain equidistant.

Note that duplicated values among the quantiles are discarded and that the minimum and maximum are replaced with `lower` and `upper`, respectively, if these are specified.

The (weighted) quantiles are computed with the function `quantileWt`.

Value

A numeric vector of breakpoints.

Author(s)

Andreas Alfons

See Also

`getCat`, `quantileWt`

Examples

```
data(eusilcS)

## semi-continuous variable, positive breakpoints equidistant
getBreaks(eusilcS$netIncome, weights=eusilcS$rb050)

## semi-continuous variable, positive breakpoints not equidistant
getBreaks(eusilcS$netIncome, weights=eusilcS$rb050,
    equidist = FALSE)
```

`getCat` *Categorize (semi-)continuous variables*

Description

Categorize continuous or semi-continuous variables. This is a utility function that is useful for writing custom wrapper functions such as `simEUSILC`.

Usage

```
getCat(x, breaks, zeros = TRUE, right = FALSE)
```

Arguments

`x`	a numeric vector to be categorized.
`breaks`	a numeric vector of two or more breakpoints.
`zeros`	a logical indicating whether `x` is semi-continuous, i.e., contains a considerable amount of zeros. See "Details" on how this affects the behavior of the function.
`right`	logical; if `zeros` is not `TRUE`, this indicates whether the intervals should be closed on the right (and open on the left) or vice versa.

Details

If `zeros` is `TRUE`, 0 is added to the breakpoints and treated as its own factor level. Consequently, intervals for negative values are left-closed and right-open, whereas intervals for positive values are left-open and right-closed.

Value

A `factor` containing the categories.

Author(s)

Andreas Alfons

See Also

`getBreaks`, `cut`

Examples

```
data(eusilcS)

## semi-continuous variable
breaks <- getBreaks(eusilcS$netIncome,
    weights=eusilcS$rb050, equidist = FALSE)
netIncomeCat <- getCat(eusilcS$netIncome, breaks)
summary(netIncomeCat)
```

`quantileWt` *Weighted sample quantiles*

Description

Compute quantiles taking into account sample weights.

Usage

```
quantileWt(x, weights = NULL, probs = seq(0, 1, 0.25), na.rm = TRUE)
```

Arguments

`x`	a numeric vector.
`weights`	an optional numeric vector containing sample weights.
`probs`	a numeric vector of probabilities with values in $[0, 1]$.
`na.rm`	a logical indicating whether any `NA` or `NaN` values should be removed from `x` before the quantiles are computed. Note that the default is `TRUE`, contrary to the function `quantile`.

Details

If weights are not specified then `quantile(x, probs, na.rm=na.rm, names=FALSE, type=1)` is used for the computation.

Note probabilities outside $[0, 1]$ cause an error.

Value

A vector of the (weighted) sample quantiles.

Author(s)

Stefan Kraft

A basic version of this function was provided by Cedric Beguin and Beat Hulliger.

See Also

`quantile`

Examples

```
data(eusilcS)
quantileWt(eusilcS$netIncome, weights=eusilcS$rb050)
```

`simCategorical` *Simulate categorical variables of population data*

Description

Simulate categorical variables of population data. The household structure of the population data needs to be simulated beforehand.

Usage

```
simCategorical(dataS, dataP, w = "rb050", strata = "db040",
               basic, additional = c("pl030", "pb220a"),
               method = c("multinom", "distribution"),
               maxit = 500, MaxNWts = 1500, seed)
```

Arguments

`dataS`	a `data.frame` containing household survey data.
`dataP`	a `data.frame` containing the simulated population household structure.
`w`	a character string specifying the column of `dataS` that contains the (personal) sample weights.
`strata`	a character string specifying the columns of `dataS` and `dataP`, respectively, that define strata. The values are simulated for each stratum separately. Note that this is currently a required argument and only one stratification variable is supported.
`basic`	a character vector specifying the columns of `dataS` and `dataP`, respectively, that define the household structure, typically age, gender and household size. The default value is `c("age", "rb090", "hsize")` if `method` is `"multinom"`, and `c("age", "rb090")` if `method` is `"distribution"`.
`additional`	a character vector specifying additional categorical variables of `dataS` that should be simulated for the population data.
`method`	a character string specifying the method to be used for simulating the additional categorical variables. Accepted values are `"multinom"` (estimation of the conditional probabilities using multinomial log-linear models and random draws from the resulting distributions), or `"distribution"` (random draws from the observed conditional distributions of their multivariate realizations).
`maxit, MaxNWts`	control parameters to be passed to `multinom` and `nnet`. See the help file for `nnet`.
`seed`	optional; an integer value to be used as the seed of the random number generator, or an integer vector containing the state of the random number generator to be restored.

Value

A `data.frame` containing the simulated population data including the categorical variables specified by `additional`.

Note

The basic household structure needs to be simulated beforehand with the function `simStructure`.

Author(s)

Andreas Alfons and Stefan Kraft

See Also

`simStructure`, `simContinuous`, `simComponents`, `simEUSILC`

Examples

```
## Not run:

## these take some time and are not run automatically
## copy & paste to the R command line

set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
eusilcP <- simStructure(eusilcS)
eusilcP <- simCategorical(eusilcS, eusilcP)
summary(eusilcP)

## End(Not run)
```

`simComponents` *Simulate components of continuous variables of population data*

Description

Simulate components of continuous variables of population data by resampling fractions from survey data. The continuous variable to be split and any categorical conditioning variables need to be simulated beforehand.

Usage

```
simComponents(dataS, dataP, w = "rb050", total = "netIncome",
              components = c("py010n", "py050n", "py090n",
                "py100n", "py110n", "py120n", "py130n", "py140n"),
              conditional = c(getCatName(total), "pl030"),
              replaceEmpty = c("sequential", "min"), seed)
```

Arguments

dataS	a data.frame containing household survey data.
dataP	a data.frame containing the simulated population data.
w	a character string specifying the column of dataS that contains the (personal) sample weights, which are used as probability weights for resampling.
total	a character string specifying the continuous variable of dataP that should be split into components. Currently, only one variable can be split at a time.
components	a character vector specifying the components in dataS that should be simulated for the population data.
conditional	an optional character vector specifying categorical contitioning variables for resampling. The fractions occurring in dataS are then drawn from the respective subsets defined by these variables.
replaceEmpty	a character string; if conditional specifies at least two conditioning variables, this determines how replacement cells for empty subsets in the sample are obtained. If "sequential", the conditioning variables are browsed sequentially such that replacement cells have the same value in one conditioning variable and minimum Manhattan distance in the other conditioning variables. If no such cells exist, replacement cells with minimum overall Manhattan distance are selected. The latter is always done if this is "min" or only one conditioning variable is used.
seed	optional; an integer value to be used as the seed of the random number generator, or an integer vector containing the state of the random number generator to be restored.

Value

A data.frame containing the simulated population data including the components of the continuous variable specified by total.

Note

The basic household structure, any categorical conditioning variables and the continuous variable to be split need to be simulated beforehand with the functions simStructure, simCategorical and simContinuous.

Author(s)

Stefan Kraft and Andreas Alfons

See Also

simStructure, simCategorical, simContinuous, simEUSILC

Examples

```
## Not run:

## these take some time and are not run automatically
## copy & paste to the R command line

set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
eusilcP <- simStructure(eusilcS)
eusilcP <- simCategorical(eusilcS, eusilcP)
basic <- c("age", "rb090", "hsize", "pl030", "pb220a")
eusilcP <- simContinuous(eusilcS, eusilcP,
    basic = basic, upper = 200000, equidist = FALSE)
# categorize net income for use as conditioning variable
breaks <- getBreaks(eusilcS$netIncome, eusilcS$rb050,
    upper=Inf, equidist = FALSE)
eusilcS$netIncomeCat <- getCat(eusilcS$netIncome, breaks)
eusilcP$netIncomeCat <- getCat(eusilcP$netIncome, breaks)
# simulate net income components
eusilcP <- simComponents(eusilcS, eusilcP)
summary(eusilcP)

## End(Not run)
```

`simContinuous` *Simulate continuous variables of population data*

Description

Simulate continuous variables of population data using multinomial log-linear models combined with random draws from the resulting categories or (two-step) regression models combined with random error terms. The household structure of the population data and any other categorical predictors need to be simulated beforehand.

Usage

```
simContinuous(dataS, dataP, w = "rb050", strata = "db040",
              basic = c("age", "rb090", "hsize"),
              additional = "netIncome",
              method = c("multinom", "lm"), zeros = TRUE,
              breaks = NULL, lower = NULL, upper = NULL,
              equidist = TRUE, gpd = TRUE, threshold = NULL,
              est = "moments", censor = NULL, log = TRUE,
              const = NULL, alpha = 0.01, residuals = TRUE,
              keep = TRUE, maxit = 500, MaxNWts = 1500,
              tol = .Machine$double.eps^0.5, seed)
```

Arguments

dataS	a data.frame containing household survey data.
dataP	a data.frame containing the simulated population data. Household structure and any other categorical predictors need to be simulated beforehand.
w	a character string specifying the column of dataS that contains the (personal) sample weights.
strata	a character string specifying the columns of dataS and dataP, respectively, that define strata. The regression models are computed for each stratum separately. Note that this is currently a required argument and only one stratification variable is supported.
basic	a character vector specifying the columns of dataS and dataP, respectively, that define the household structure and any other categorical predictors, such as age, gender and household size.
additional	a character string specifying the additional continuous variable of dataS that should be simulated for the population data. Currently, only one additional variable can be simulated at a time.
method	a character string specifying the method to be used for simulating the continuous variable. Accepted values are "multinom", for using multinomial log-linear models combined with random draws from the resulting ategories, and "lm", for using (two-step) regression models combined with random error terms.
zeros	a logical indicating whether the variable specified by additional is semi-continuous, i.e., contains a considerable amount of zeros. If TRUE and method is "multinom", a separate factor level for zeros in the response is used. If TRUE and method is "lm", a two-step model is applied. The first step thereby uses a log-linear or multinomial log-linear model (see "Details").
breaks	an optional numeric vector; if multinomial models are computed, this can be used to supply two or more breakpoints for categorizing the variable specified by additional. If NULL, breakpoints are computed using weighted quantiles.
lower, upper	optional numeric values; if multinomial models are computed and breaks is NULL, these can be used to specify lower and upper bounds other than minimum and maximum, respectively. Note that if method is "multinom" and gpd is TRUE (see below), upper defaults to Inf.
equidist	logical; if method is "multinom" and breaks is NULL, this indicates whether the (positive) default breakpoints should be equidistant or whether there should be refinements in the lower and upper tail (see getBreaks).
gpd	logical; if method is "multinom", this indicates whether the upper tail of the variable specified by additional should be simulated by random draws from a (truncated) generalized Pareto distribution rather than a uniform distribution.
threshold	a numeric value; if method is "multinom", values for categories above threshold are drawn from a (truncated) generalized Pareto distribution.
est	a character string; if method is "multinom", the estimator to be used to fit the generalized Pareto distribution (see fitgpd).
censor	an optional named list of data.frames; if multinomial models are computed, this can be used to account for structural zeros. The names of the list components

specify the categories that should be censored. For each of these categories, a `data.frame` containing levels of the predictor variables can be supplied. The probability of the specified categories is set to 0 for the respective predictor levels. Currently, this is only implemented for more than two categories in the response.

`log` logical; if `method` is `"lm"`, this indicates whether the linear model should be fitted to the logarithms of the variable specified by `additional`. The predicted values are then back-transformed with the exponential function. See "Details" for more information.

`const` numeric; if `method` is `"lm"` and `log` is `TRUE`, this gives a constant to be added before log transformation.

`alpha` numeric; if `method` is `"lm"`, this gives trimming parameters for the sample data. Trimming is thereby done with respect to the variable specified by `additional`. If a numeric vector of length two is supplied, the first element gives the trimming proportion for the lower part and the second element the trimming proportion for the upper part. If a single numeric is supplied, it is used for both. With `NULL`, trimming is suppressed.

`residuals` logical; if `method` is `"lm"`, this indicates whether the random error terms should be obtained by draws from the residuals. If `FALSE`, they are drawn from a normal distribution (median and MAD of the residuals are used as parameters).

`keep` logical; if multinomial models are computed, this indicates whether the simulated categories should be stored as a variable in the resulting population data. If `TRUE`, the corresponding column name is given by `additional` with postfix `"Cat"`.

`maxit, MaxNWts`
control parameters to be passed to `multinom` and `nnet`. See the help file for `nnet`.

`tol` if `method` is `"lm"`, a small positive numeric value or `NULL`. When fitting a log-linear model within a stratum, factor levels may not exist in the sample but are likely to exist in the population. However, the coefficient for such factor levels will be 0. Therefore, coefficients smaller than `tol` in absolute value are replaced by coefficients from an auxiliary model that is fit to the whole sample. If `NULL`, no auxiliary log-linear model is computed and no coefficients are replaced.

`seed` optional; an integer value to be used as the seed of the random number generator, or an integer vector containing the state of the random number generator to be restored.

Details

If `method` is `"lm"`, the behavior for two-step models is described in the following.

If `zeros` is `TRUE` and `log` is not `TRUE` or the variable specified by `additional` does not contain negative values, a log-linear model is used to predict whether an observation is zero or not. Then a linear model is used to predict the non-zero values.

If `zeros` is `TRUE`, `log` is `TRUE` and `const` is specified, again a log-linear model is used to predict whether an observation is zero or not. In the linear model to predict the non-zero values, `const` is added to the variable specified by `additional` before the logarithms are taken.

If `zeros` is `TRUE`, `log` is `TRUE`, `const` is `NULL` and there are negative values, a multinomial log-linear model is used to predict negative, zero and positive observations. Categories for the negative values are thereby defined by `breaks`. In the second step, a linear model is used to predict the positive values and negative values are drawn from uniform distributions in the respective classes.

If `zeros` is `FALSE`, `log` is `TRUE` and `const` is `NULL`, a two-step model is used if there are non-positive values in the variable specified by `additional`. Whether a log-linear or a multinomial log-linear model is used depends on the number of categories to be used for the non-positive values, as defined by `breaks`. Again, positive values are then predicted with a linear model and non-positive values are drawn from uniform distributions.

Value

A `data.frame` containing the simulated population data including the continuous variable specified by `additional`.

Note

The basic household structure and any other categorical predictors need to be simulated beforehand with the functions `simStructure` and `simCategorical`, respectively.

Author(s)

Original code by Stefan Kraft, redesign and generalizations by Andreas Alfons.

See Also

`simStructure`, `simCategorical`, `simComponents`, `simEUSILC`

Examples

```
## Not run:

## these take some time and are not run automatically
## copy & paste to the R command line

set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
eusilcP <- simStructure(eusilcS)
eusilcP <- simCategorical(eusilcS, eusilcP)
basic <- c("age", "rb090", "hsize", "pl030", "pb220a")

# multinomial model with random draws
eusilcM <- simContinuous(eusilcS, eusilcP,
    basic = basic, upper = 200000, equidist = FALSE)
summary(eusilcM)

# two-step regression
eusilcT <- simContinuous(eusilcS, eusilcP,
    basic = basic, method = "lm")
summary(eusilcT)
```

```
## End(Not run)
```

`simEUSILC` *Simulate EU-SILC population data*

Description

Simulate population data for the European Statistics on Income and Living Conditions (EU-SILC).

Usage

```
simEUSILC(dataS, hid = "db030", wh = "db090", wp = "rb050",
          hsize = NULL, strata = "db040", pid = NULL, age = "age",
          gender = "rb090", categorizeAge = TRUE, breaksAge = NULL,
          categorical = c("pl030", "pb220a"),
          income = "netIncome", method = c("multinom", "twostep"),
          breaks = NULL, lower = NULL, upper = NULL,
          equidist = TRUE, gpd = TRUE, threshold = NULL,
          est = "moments", const = NULL, alpha = 0.01,
          residuals = TRUE,
          components = c("py010n", "py050n", "py090n",
            "py100n", "py110n", "py120n", "py130n", "py140n"),
          conditional = c(getCatName(income), "pl030"),
          keep = TRUE, maxit = 500, MaxNWts = 1500,
          tol = .Machine$double.eps^0.5, seed)
```

Arguments

`dataS`	a `data.frame` containing EU-SILC survey data.
`hid`	a character string specifying the column of `dataS` that contains the household ID.
`wh`	a character string specifying the column of `dataS` that contains the household sample weights.
`wp`	a character string specifying the column of `dataS` that contains the personal sample weights.
`hsize`	an optional character string specifying a column of `dataS` that contains the household size. If `NULL`, the household sizes are computed.
`strata`	a character string specifying the column of `dataS` that define strata. Note that this is currently a required argument and only one stratification variable is supported.
`pid`	an optional character string specifying a column of `dataS` that contains the personal ID.
`age`	a character string specifying the column of `dataS` that contains the age of the persons (to be used for setting up the household structure).

gender	a character string specifying the column of dataS that contains the gender of the persons (to be used for setting up the household structure).
categorizeAge	a logical indicating whether age categories should be used for simulating additional categorical and continuous variables to decrease computation time.
breaksAge	numeric; if categorizeAge is TRUE, an optional vector of two or more breakpoints for constructing age categories, otherwise ignored.
categorical	a character vector specifying additional categorical variables of dataS that should be simulated for the population data.
income	a character string specifying the variable of dataS that contains the personal income (to be simulated for the population data).
method	a character string specifying the method to be used for simulating personal income. Accepted values are "multinom" (for using multinomial log-linear models combined with random draws from the resulting ategories) and "twostep" (for using two-step regression models combined with random error terms).
breaks	if method is "multinom", an optional numeric vector of two or more breakpoints for categorizing the personal income. If missing, breakpoints are computed using weighted quantiles.
lower, upper	numeric values; if method is "multinom" and breaks is NULL, these can be used to specify lower and upper bounds other than minimum and maximum, respectively. Note that if gpd is TRUE (see below), upper defaults to Inf.
equidist	logical; if method is "multinom" and breaks is NULL, this indicates whether the (positive) default breakpoints should be equidistant or whether there should be refinements in the lower and upper tail (see getBreaks).
gpd	logical; if method is "multinom", this indicates whether the upper tail of the personal income should be simulated by random draws from a (truncated) generalized Pareto distribution rather than a uniform distribution.
threshold	a numeric value; if method is "multinom", values for categories above threshold are drawn from a (truncated) generalized Pareto distribution.
est	a character string; if method is "multinom", the estimator to be used to fit the generalized Pareto distribution (see fitgpd).
const	numeric; if method is "twostep", this gives a constant to be added before log transformation.
alpha	numeric; if method is "twostep", this gives trimming parameters for the sample data. Trimming is thereby done with respect to the variable specified by additional. If a numeric vector of length two is supplied, the first element gives the trimming proportion for the lower part and the second element the trimming proportion for the upper part. If a single numeric is supplied, it is used for both. With NULL, trimming is suppressed.
residuals	logical; if method is "twostep", this indicates whether the random error terms should be obtained by draws from the residuals. If FALSE, they are drawn from a normal distribution (median and MAD of the residuals are used as parameters).
components	a character vector specifying the income components in dataS (to be simulated for the population data).

`conditional`	an optional character vector specifying categorical contitioning variables for re-sampling of the income components. The fractions occurring in `dataS` are then drawn from the respective subsets defined by these variables.
`keep`	a logical indicating whether variables computed internally in the procedure (such as the original IDs of the corresponding households in the underlying sample, age categories or income categories) should be stored in the resulting population data.
`maxit, MaxNWts`	control parameters to be passed to `multinom` and `nnet`. See the help file for `nnet`.
`tol`	if `method` is `"twostep"`, a small positive numeric value or `NULL` (see `simContinuous`).
`seed`	optional; an integer value to be used as the seed of the random number generator, or an integer vector containing the state of the random number generator to be restored.

Value

A `data.frame` containing the simulated EU-SILC population data.

Note

This is a wrapper calling `simStructure`, `simCategorical`, `simContinuous` and `simComponents`.

Author(s)

Andreas Alfons and Stefan Kraft

See Also

`simStructure`, `simCategorical`, `simContinuous`, `simComponents`

Examples

```
## Not run:

## these take some time and are not run automatically
## copy & paste to the R command line

set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data

# multinomial model with random draws
eusilcM <- simEUSILC(eusilcS, upper = 200000, equidist = FALSE)
summary(eusilcM)

# two-step regression
eusilcT <- simEUSILC(eusilcS, method = "twostep")
summary(eusilcT)

## End(Not run)
```

`simStructure` *Simulate the household structure of population data*

Description

Simulate basic categorical variables that define the household structure (typically household ID, age and gender) of population data by resampling from survey data.

Usage

```
simStructure(dataS, hid = "db030", w = "db090",
             hsize = NULL, strata = "db040", pid = NULL,
             additional = c("age", "rb090"),
             method = c("direct", "multinom", "distribution"),
             keep = TRUE, seed)
```

Arguments

`dataS`	a `data.frame` containing household survey data.
`hid`	a character string specifying the column of `dataS` that contains the household ID.
`w`	a character string specifying the column of `dataS` that contains the (household) sample weights, which are used as probability weights for resampling.
`hsize`	an optional character string specifying a column of `dataS` that contains the household size. If `NULL`, the household sizes are computed.
`strata`	a character string specifying the column of `dataS` that define strata. Note that this is currently a required argument and only one stratification variable is supported.
`pid`	an optional character string specifying a column of `dataS` that contains the personal ID.
`additional`	a character vector specifying additional categorical variables of `dataS` that define the household structure, typically age and gender.
`method`	a character string specifying the method to be used for simulating the household sizes. Accepted values are `"direct"` (estimation of the population totals for each combination of stratum and household size using the Horvitz-Thompson estimator), `"multinom"` (estimation of the conditional probabilities within the strata using a multinomial log-linear model and random draws from the resulting distributions), or `"distribution"` (random draws from the observed conditional distributions within the strata).
`keep`	a logical indicating whether the original IDs of the corresponding households in the underlying sample should be stored as a variable in the resulting population data. If `TRUE`, the corresponding column name is given by `hid` with postfix `"Sample"`.
`seed`	optional; an integer value to be used as the seed of the random number generator, or an integer vector containing the state of the random number generator to be restored.

Value

A `data.frame` containing the simulated population household structure.

Note

The function `sample` is used, which gives results incompatible with those from R < 2.2.0 and produces a warning the first time this happens in a session.

Author(s)

Andreas Alfons and Stefan Kraft

See Also

`simCategorical`, `simContinuous`, `simComponents`, `simEUSILC`

Examples

```
set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
eusilcP <- simStructure(eusilcS)
summary(eusilcP)
```

---

spBwplot *Weighted box plots*

Description

Produce box-and-whisker plots of continuous or semi-continuous variables, possibly broken down according to conditioning variables and taking into account sample weights.

Usage

```
spBwplot(x, ...)

## Default S3 method:
spBwplot(x, weights = NULL, cond = NULL, dataS, dataP = NULL,
         horizontal = TRUE, coef = 1.5, zeros = TRUE,
         minRatio = NULL, do.out = FALSE, ...)
```

Arguments

`x`	for the default method (currently the only method implemented), a character vector specifying the columns of `dataS` and `dataP` to be plotted.
`weights`	a character string specifying the column of `dataS` that contains the (personal) sample weights.
`cond`	an optional character vector specifying conditioning variables.

dataS	a data.frame containing household survey data.
dataP	optional; a data.frame containing simulated population data or a list of such data.frames.
horizontal	a logical indicating whether the boxes should be horizontal or vertical.
coef	a numeric value that determines the extension of the whiskers.
zeros	a logical indicating whether the variables specified by x are semi-continuous, i.e., contain a considerable amount of zeros. If TRUE, the box widths correspond to the proportion of non-zero data points and the (weighted) box plot statistics are computed for these non-zero data points only.
minRatio	a numeric value in $(0, 1]$; if zeros is TRUE, the boxes may become unreadable for a large proportion of zeros. In such a case, this can be used to specify a minimum ratio for the box widths. Variable box widths for semi-continuous variables can be suppressed by setting this value to 1.
do.out	a logical indicating whether data points that lie beyond the extremes of the whiskers should be plotted. Note that this is FALSE by default.
...	for the generic function, further arguments to be passed down to methods. For the default method, further arguments to be passed to bwplot.

Details

Missing values are ignored for producing box plots.

Value

An object of class "trellis", as returned by bwplot.

Note

A formula interface may be added in the future.

Author(s)

Andreas Alfons

See Also

spBwplotStats, bwplot

Examples

```
## Not run:

## these take some time and are not run automatically
## copy & paste to the R command line

set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
# multinomial model with random draws
```

```
eusilcM <- simEUSILC(eusilcS, upper = 200000, equidist = FALSE)
# two-step regression
eusilcT <- simEUSILC(eusilcS, method = "twostep")
# plot results by gender
spBwplot("netIncome", "rb050", "rb090", dataS = eusilcS,
    dataP = list(M = eusilcM, T = eusilcT), layout = c(1, 2))

## End(Not run)
```

`spBwplotStats` *Weighted box plot statistics*

Description

Compute the statistics necessary for producing box-and-whisker plots of continuous or semi-continuous variables, taking into account sample weights.

Usage

```
spBwplotStats(x, weights = NULL, coef = 1.5,
              zeros = TRUE, do.out = TRUE)
```

Arguments

`x`	a numeric vector.
`weights`	an optional numeric vector containing sample weights.
`coef`	a numeric value that determines the extension of the whiskers.
`zeros`	a logical indicating whether the variable specified by `additional` is semi-continuous, i.e., contains a considerable amount of zeros. If `TRUE`, the (weighted) box plot statistics are computed for the non-zero data points only and the number of zeros is returned, too.
`do.out`	a logical indicating whether data points that lie beyond the extremes of the whiskers should be returned.

Details

The function `quantileWt` is used for the computation of (weighted) quantiles. The median is computed together with the first and the third quartile, which form the box. If `range` is positive, the whiskers extend to the most extreme data points that have a distance to the box of no more than `coef` times the interquartile range. For `coef = 0`, the whiskers mark the minimum and the maximum of the sample, whereas a negative value causes an error.

Value

A list of class "spBwplotStats" with the following components:

stats	A vector of length 5 containing the (weighted) statistics for the construction of a box plot.
n	if weights is NULL, the number of non-missing and, if zeros is TRUE, non-zero data points. Otherwise the sum of the weights of the corresponding points.
nzero	if zeros is TRUE and weights is NULL, the number of zeros. If zeros is TRUE and weights is not NULL, the sum of the weights of the zeros. If zeros is not TRUE, this is NULL.
out	if do.out, the values of any data points that lie beyond the extremes of the whiskers.

Author(s)

Stefan Kraft and Andreas Alfons

See Also

spBwplot, for producing (weighted) box plots of continuous or semi-continuous variables.

quantileWt for the computation of (weighted) sample quantiles.

boxplot.stats for the unweighted statistics for box plots (not considering semi-continuous variables).

Examples

```
data(eusilcS)

## semi-continuous variable
spBwplotStats(eusilcS$netIncome,
    weights=eusilcS$rb050, do.out = FALSE)
```

---

spCdf *(Weighted empirical) cumulative distribution function*

Description

Compute a (weighted empirical) cumulative distribution function for survey or population data. For survey data, sample weights are taken into account.

Usage

```
spCdf(x, weights = NULL, approx = FALSE, n = 10000)
```

Arguments

`x`	a numeric vector.
`weights`	an optional numeric vector containing sample weights.
`approx`	a logical indicating whether an approximation of the cumulative distribution function should be computed.
`n`	a single integer value; if `approx` is `TRUE`, this specifies the number of points at which the approximation takes place (see `approx`).

Details

Sample weights are taken into account by adjusting the step height. To be precise, the weighted step height for an observation is defined as its weight divided by the sum of all weights $(w_i / \sum_{j=1}^{n} w_j)$.

If requested, the approximation is performed using the function `approx`.

Value

A list of class `"spCdf"` with the following components:

`x`	a numeric vector containing the x-coordinates.
`y`	a numeric vector containing the y-coordinates.
`approx`	a logical indicating whether the coordinates represent an approximation.

Author(s)

Andreas Alfons and Stefan Kraft

See Also

`spCdfplot`, `ecdf`, `approx`

Examples

```
data(eusilcS)
cdfS <- spCdf(eusilcS$netIncome, weights = eusilcS$rb050)
plot(cdfS, type="s")
```

`spCdfplot` *Plot (weighted empirical) cumulative distribution functions*

Description

Plot (weighted empirical) cumulative distribution functions for survey and population data, possibly broken down according to conditioning variables. For survey data, sample weights are taken into account.

Usage

```
spCdfplot(x, ...)

## Default S3 method:
spCdfplot(x, weights = NULL, cond = NULL, dataS, dataP = NULL,
          approx = NULL, n = 10000, bounds = TRUE, ...)
```

Arguments

x	for the default method (currently the only method implemented), a character vector specifying the columns of dataS and dataP to be plotted.
weights	a character string specifying the column of dataS that contains the (personal) sample weights.
cond	an optional character vector specifying conditioning variables.
dataS	a data.frame containing household survey data.
dataP	optional; a data.frame containing simulated population data or a list of such data.frames.
approx	logicals indicating whether approximations of the cumulative distribution functions should be computed. The default is to use FALSE for the survey data and TRUE for any population data. If no population data are supplied, a single logical should be used. On the other hand, if any population data are supplied, the behavior is as follows. If a single logical is supplied, it is used for the population data and FALSE is used for the survey data. If a vector of length two is supplied, the first value is used for the survey data and the second for the population data. Note that if multiple populations are supplied, the same value is used for all of them.
n	integers specifying the number of points at which the approximations for the respective data sets take place (see approx). If a single value is supplied, it is used wherever approx is TRUE. If a vector of length two and any population data are supplied, the first value is used for the survey data and the second for the population data (in case the corresponding values of approx are TRUE). Note that if multiple populations are supplied, the same value is used for all of them.
bounds	a logical indicating whether vertical lines should be drawn at 0 and 1 (the bounds for cumulative distribution functions).
...	for the generic function, further arguments to be passed down to methods. For the default method, further arguments to be passed to xyplot.

Details

Sample weights are taken into account by adjusting the step height. To be precise, the weighted step height for an observation is defined as its weight divided by the sum of all weights $(w_i / \sum_{j=1}^{n} w_j)$.

Value

An object of class "trellis", as returned by xyplot.

Note

A formula interface may be added in the future.

Author(s)

Andreas Alfons

See Also

spCdf, xyplot

Examples

```
## Not run:

## these take some time and are not run automatically
## copy & paste to the R command line

set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
# multinomial model with random draws
eusilcM <- simEUSILC(eusilcS, upper = 200000, equidist = FALSE)
# two-step regression
eusilcT <- simEUSILC(eusilcS, method = "twostep")
# plot results by gender
spCdfplot("netIncome", "rb050", "rb090", dataS = eusilcS,
    dataP = list(M = eusilcM, T = eusilcT), layout = c(1, 2))

## End(Not run)
```

`spMosaic` *Mosaic plots of expected and realized population sizes*

Description

Create mosaic plots of expected (i.e., estimated) and realized (i.e., simulated) population sizes.

Usage

```
spMosaic(x, ...)

## Default S3 method:
spMosaic(x, weights = NULL, dataS, dataP, ...)
```

Arguments

`x`	for the default method, an optional character vector specifying the columns of `dataS` and `dataP` to be cross tabulated and plotted. Otherwise, an object of class `"spTable"`.
`weights`	either a numeric vector containing the (personal) sample weights, or a character string specifying the corresponding column of `dataS`.
`dataS`	a `data.frame` containing household survey data.
`dataP`	a `data.frame` containing simulated population data.
`...`	further arguments to be passed to `cotabplot`.

Details

The two tables of expected and realized population sizes are combined into a single table, with an additional contitioning variable indicating expected and realized values. A conditional plot of this table is then produced using `cotabplot`.

Note

A formula interface may be added in the future.

Author(s)

Andreas Alfons

See Also

`spTable`, `cotabplot`

Examples

```
set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
eusilcP <- simStructure(eusilcS)
abb <- c("B","LA","Vi","C","St","UA","Sa","T","Vo")
spMosaic(c("rb090", "db040", "hsize"), "rb050", eusilcS, eusilcP,
    labeling=labeling_border(abbreviate=c(db040=TRUE)))
```

`spTable` *Cross tabulations of expected and realized population sizes*

Description

Compute contingency tables of expected (i.e., estimated) and realized (i.e., simulated) population sizes. The expected values are obtained with the Horvitz-Thompson estimator.

Usage

```
spTable(dataS, dataP, select = NULL, weights = NULL)

## S3 method for class 'formula':
spTable(dataS, dataP, select, weights = NULL)
```

Arguments

`dataS`	a `data.frame` containing household survey data.
`dataP`	a `data.frame` containing simulated population data.
`select`	for the formula method, a formula specifying the variables to be used for cross tabulation. For the default method, an optional character vector defining the columns of `dataS` and `dataP` to be used.
`weights`	either a numeric vector containing the (personal) sample weights, or the name of the corresponding column of `dataS` (for the default method, the name must be a character string).

Details

The contingency tables are computed with `tableWt`.

Value

A list of class `"spTable"` with the following components:

`expected`	the contingency table estimated from the survey data.
`realized`	the contingency table computed from the simulated population data.

Note

The class of the argument `select` determines the method to be dispatched, not the class of the first argument.

Author(s)

Andreas Alfons

See Also

`spMosaic`, `tableWt`

Examples

```
set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
eusilcP <- simStructure(eusilcS)
spTable(eusilcS, eusilcP, select = ~ rb090 + db040, weights = rb050)
```

`tableWt` *Weighted cross tabulation*

Description

Compute contingency tables taking into account sample weights.

Usage

```
tableWt(x, weights = NULL, useNA = c("no", "ifany", "always"))
```

Arguments

`x`	a vector that can be interpreted as a factor, or a matrix or `data.frame` whose columns can be interpreted as factors.
`weights`	an optional numeric vector containing sample weights.
`useNA`	a logical indicating whether to include extra `NA` levels in the table.

Details

For each combination of the variables in `x`, the weighted number of occurence is computed as the sum of the corresponding sample weights. If weights are not specified, the function `table` is applied.

Value

The (weighted) contingency table as an object of class `table`, an array of integer values.

Author(s)

Andreas Alfons and Stefan Kraft

See Also

`table`, `contingencyWt`

Examples

```
data(eusilcS)
tableWt(eusilcS[, c("hsize", "db040")], weights = eusilcS$rb050)
tableWt(eusilcS[, c("rb090", "pb220a")], weights = eusilcS$rb050,
    useNA = "ifany")
```

`utils` *Weighted mean, variance, covariance matrix and correlation matrix*

Description

Compute mean, variance, covariance matrix and correlation matrix, taking into account sample weights.

Usage

```
meanWt(x, weights, na.rm = TRUE)

varWt(x, weights, na.rm = TRUE)

covWt(x, ...)

## Default S3 method:
covWt(x, y, weights, ...)

## S3 method for class 'matrix':
covWt(x, weights, ...)

## S3 method for class 'data.frame':
covWt(x, weights, ...)

corWt(x, ...)

## Default S3 method:
corWt(x, y, weights, ...)

## S3 method for class 'matrix':
corWt(x, weights, ...)

## S3 method for class 'data.frame':
corWt(x, weights, ...)
```

Arguments

`x`	for `meanWt` and `varWt`, a numeric vector. For `covWt` and `corWt`, a numeric vector, matrix or `data.frame`.
`y`	a numeric vector. If missing, this defaults to `x`.
`weights`	an optional numeric vector containing sample weights.
`na.rm`	a logical indicating whether any `NA` or `NaN` values should be removed from `x` before computation. Note that the default is `TRUE`.
`...`	for the generic functions `covWt` and `corWt`, additional arguments to be passed to methods. Additional arguments not included in the definition of the methods are ignored.

Details

`meanWt` is a simple wrapper that calls `mean(x, na.rm=na.rm)` if `weights` is missing and `weighted.mean(x, w=weights, na.rm=na.rm)` otherwise.

`varWt` calls `var(x, na.rm=na.rm)` if `weights` is missing.

`covWt` and `corWt` always remove missing values pairwise and call `cov` and `cor`, respectively, if `weights` is missing.

Value

For `meanWt`, the (weighted) mean.

For `varWt`, the (weighted) variance.

For `covWt`, the (weighted) covariance matrix or, for the default method, the (weighted) covariance.

For `corWt`, the (weighted) correlation matrix or, for the default method, the (weighted) correlation coefficient.

Author(s)

Stefan Kraft and Andreas Alfons

See Also

`mean`, `weighted.mean`, `var`, `cov`, `cor`

Examples

```
data(eusilcS)
meanWt(eusilcS$netIncome, weights=eusilcS$rb050)
sqrt(varWt(eusilcS$netIncome, weights=eusilcS$rb050))
```

Index

∗Topic **array**
utils, 30
∗Topic **category**
contingencyWt, 3
tableWt, 29
∗Topic **datagen**
simCategorical, 9
simComponents, 10
simContinuous, 12
simEUSILC, 16
simStructure, 19
∗Topic **datasets**
eusilcS, 4
∗Topic **dplot**
spBwplotStats, 22
spCdf, 23
spTable, 27
∗Topic **hplot**
spBwplot, 20
spCdfplot, 24
spMosaic, 26
∗Topic **manip**
getBreaks, 6
getCat, 7
∗Topic **multivariate**
utils, 30
∗Topic **package**
simPopulation-package, 1
∗Topic **univar**
quantileWt, 8
utils, 30

approx, *24, 25*

boxplot.stats, *23*
bwplot, *21*

contingencyWt, 3, *29*
cor, *31*
corWt *(utils)*, 30
cotabplot, *27*
cov, *31*
covWt *(utils)*, 30
cut, *8*

ecdf, *24*
eusilcS, 4

factor, *7*
fitgpd, *13*, *17*

getBreaks, 6, *8*, *13*, *17*
getCat, 7, *7*

mean, *31*
meanWt *(utils)*, 30
multinom, *10*, *14*, *18*

nnet, *10*, *14*, *18*

quantile, *8, 9*
quantileWt, *6, 7*, 8, *22, 23*

sample, *20*
simCategorical, 9, *11, 12*, *15*, *18*, *20*
simComponents, 10, *10*, *15*, *18*, *20*
simContinuous, *10, 11*, 12, *12*, *18*, *20*
simEUSILC, *6, 7*, *10*, *12*, *15*, 16, *20*
simPopulation
(simPopulation-package), 1
simPopulation-package, 1
simStructure, *10–12*, *15*, *18*, 19
spBwplot, 20, *23*
spBwplotStats, *21*, 22
spCdf, 23, *26*
spCdfplot, 24, *24*
spMosaic, 26, *28*
spTable, 27, *27*

table, *29*
tableWt, *3, 4*, *28*, 29

utils, 30

var, *31*
varWt *(utils)*, 30

weighted.mean, *31*

xyplot, *25, 26*

Package 'VIM'

May 5, 2011

Version 2.0.1

Date 2011-05-05

Title Visualization and Imputation of Missing Values

Author Matthias Templ, Andreas Alfons, Alexander Kowarik

Maintainer Matthias Templ <templ@tuwien.ac.at>

Depends R (>= 2.10),e1071,car, colorspace, nnet, robustbase, tcltk,tkrplot, sp, vcd, Rcpp

Imports car, colorspace, grDevices, robustbase, stats, tcltk, sp,utils, vcd

Description This package introduces new tools for the visualization of missing values in R, which can be used for exploring the data and the structure of the missing values. Depending on this structure, they may help to identify the mechanism generating the missings. A graphical user interface allows an easy handling of the implemented plot methods.

License GPL (>= 2)

URL http://cran.r-project.org/package=VIM

Repository CRAN

Date/Publication 2011-05-05 10:56:52

R topics documented:

VIM-package 2
aggr 3
alphablend 6
barMiss 6
bgmap 8
chorizonDL 9
colormapMiss 13
colSequence 15

count . . . 17
growdotMiss . . . 17
histMiss . . . 19
hotdeck . . . 22
initialise . . . 23
irmi . . . 23
kNN . . . 25
kola.background . . . 27
mapMiss . . . 27
marginmatrix . . . 29
marginplot . . . 30
matrixplot . . . 32
mosaicMiss . . . 34
pairsVIM . . . 35
parcoordMiss . . . 37
pbox . . . 39
prepare . . . 41
print.aggr . . . 43
print.summary.aggr . . . 44
rugNA . . . 44
SBS5242 . . . 45
scattJitt . . . 46
scattmatrixMiss . . . 47
scattMiss . . . 49
sleep . . . 51
spineMiss . . . 52
summary.aggr . . . 54
tao . . . 55
vmGUImenu . . . 56

Index **58**

`VIM-package` *Visualization and Imputation of Missing Values*

Description

This package introduces new tools for the visualization of missing values in R, which can be used for exploring the data and the structure of the missing values. Depending on this structure, they may help to identify the mechanism generating the missing values. This knowledge is necessary for selecting an appropriate imputation method in order to reliably estimate the missing values. Thus the visualization tools should be applied before imputation.

Detecting missing values mechanisms is usually done by statistical tests or models. Visualization of missing values can support the test decision, but also reveals more details about the data structure. Most notably, statistical requirements for a test can be checked graphically, and problems like outliers or skewed data distributions can be discovered. Furthermore, the included plot methods may also be able to detect missing values mechanisms in the first place.

A graphical user interface allows an easy handling of the plot methods. In addition, VIM can be used for data from essentially any field.

Robust imputation methods and diagnstic tools will be included in future versions.

Details

Package:	VIM
Type:	Package
Version:	1.4
Date:	2010-01-18
License:	GPL (>= 2)

Author(s)

Matthias Templ, Andreas Alfons, Alexander Kowarik

Maintainer: Matthias Templ <templ@statistik.tuwien.ac.at>

`aggr` *Aggregations for missing values*

Description

Calculate or plot the amount of missing values in each variable and the amount of missing values in certain combinations of variables.

Usage

```
aggr(x, plot = TRUE, ...)

## S3 method for class 'aggr'
plot(x, col = c("skyblue","red"), bars = TRUE,
     numbers = FALSE, prop = TRUE, border = par("fg"),
     sortVars = FALSE, sortCombs = TRUE, ylabs = NULL,
     axes = TRUE, labels = axes, cex.lab = 1.2,
     cex.axis = par("cex"), cex.numbers = par("cex"),
     gap = 4, ...)

TKRaggr(x, ..., hscale = NULL, vscale = NULL, TKRpar = list())
```

Arguments

`x`	a vector, matrix or `data.frame`.
`plot`	a logical indicating whether the results should be plotted (the default is `TRUE`).
`col`	a vector of length two giving the colors to be used for observed and missing data. If only one color is supplied, it is used for missing data and observed data is transparent.
`bars`	a logical indicating whether a small barplot for the frequencies of the different combinations should be drawn.
`numbers`	a logical indicating whether the proportion or frequencies of the different combinations should be represented by numbers.
`prop`	a logical indicating whether the proportion of missing values and combinations should be used rather than the total amount.
`border`	the color to be used for the border of the bars and rectangles. Use `border=NA` to omit borders.
`sortVars`	a logical indicating whether the variables should be sorted by the number of missing values.
`sortCombs`	a logical indicating whether the combinations should be sorted by the frequency of occurrence.
`ylabs`	a character vector of length two giving the y-axis labels for the two plots.
`axes`	a logical indicating whether axes should be drawn.
`labels`	either a logical indicating whether labels should be plotted on the x-axis, or a character vector giving the labels.
`cex.lab`	the character expansion factor to be used for the axis labels.
`cex.axis`	the character expansion factor to be used for the axis annotation.
`cex.numbers`	the character expansion factor to be used for the proportion or frequencies of the different combinations
`gap`	a numeric value giving the distance between the two plots in margin lines.
`...`	for `aggr` and `TKRaggr`, further arguments and graphical parameters to be passed to `plot.aggr`. For `plot.aggr`, further graphical parameters to be passed down. `par("oma")` will be set appropriately unless supplied (see `par`).
`hscale`	horizontal scale factor for plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of variables.
`vscale`	vertical scale factor for the plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of combinations.
`TKRpar`	a list of graphical parameters to be set for the plot to be embedded in a *Tcl/Tk* window (see 'Details' and `par`).

Details

Often it is of interest how many missing values are contained in each variable. Even more interesting, there may be certain combinations of variables with a high number of missing values.

The barplot on the left hand side shows the number of missing values in each variable. In the *aggregation plot* on the right hand side, all existing combinations of missing and non-missing values in the observations are visualized. Available and missing data are color coded as given by `col`. Additionally, the proportions or frequencies of the different combinations can be represented by a small bar plot and numbers. Variables may be sorted by the number of missing values and combinations by the frequency of occurrence to give more power to finding the structure of missing values.

The graphical parameter `oma` will be set unless supplied as an argument.

`TKRaggr` behaves like `plot.aggr`, but uses `tkrplot` to embed the plot in a *Tcl/Tk* window. This is useful if the number of variables and/or combinations is large, because scrollbars allow to move from one part of the plot to another.

Value

for `aggr`, a list of class `"aggr"` containing the following components:

`x`	the data used.
`combinations`	a character vector representing the combinations of variables.
`count`	the frequencies of these combinations.
`percent`	the percentage of these combinations.
`missings`	a `data.frame` containing the amount of missing values in each variable.
`tabcomb`	the indicator matrix for the combinations of variables.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `labs` and `names.arg` can still be supplied to `...` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `ylabs` and `labels` instead.

Author(s)

Andreas Alfons, Matthias Templ

See Also

`print.aggr`, `summary.aggr`

Examples

```
data(sleep, package="VIM")
a <- aggr(sleep)
a
summary(a)
```

alphablend *Alphablending for colors*

Description

Convert colors to semitransparent colors.

Usage

```
alphablend(col, alpha = NULL, bg = NULL)
```

Arguments

col	a vector specifying colors.
alpha	a numeric vector containing the alpha values (between 0 and 1).
bg	the background color to be used for alphablending. This can be used as a workaround for graphics devices that do not support semitransparent colors.

Value

a vector containing the semitransparent colors.

Author(s)

Andreas Alfons

Examples

```
alphablend("red", 0.6)
```

barMiss *Barplot with information about missing values*

Description

Barplot with highlighting of missing values in other variables by splitting each bar into two parts. Additionally, information about missing values in the variable of interest is shown on the right hand side.

Usage

```
barMiss(x, pos = 1, selection = c("any","all"),
        col = c("skyblue","red","skyblue4","red4"),
        border = NULL, main = NULL, sub = NULL,
        xlab = NULL, ylab = NULL, axes = TRUE,
        labels = axes, only.miss = TRUE,
        miss.labels = axes, interactive = TRUE, ...)
```

Arguments

`x`	a vector, matrix or `data.frame`.
`pos`	a numeric value giving the index of the variable of interest. Additional variables in `x` are used for highlighting.
`selection`	the selection method for highlighting missing values in multiple additional variables. Possible values are `"any"` (highlighting of missing values in *any* of the additional variables) and `"all"` (highlighting of missing values in *all* of the additional variables).
`col`	a vector of length four giving the colors to be used. If only one color is supplied, the bars are transparent and the supplied color is used for highlighting. Else if two colors are supplied, they are recycled.
`border`	the color to be used for the border of the bars. Use `border=NA` to omit borders.
`main, sub`	main and sub title.
`xlab, ylab`	axis labels.
`axes`	a logical indicating whether axes should be drawn on the plot.
`labels`	either a logical indicating whether labels should be plotted below each bar, or a character vector giving the labels.
`only.miss`	logical; if `TRUE`, the missing values in the variable of interest are visualized by a single bar. Otherwise, a small barplot is drawn on the right hand side (see 'Details').
`miss.labels`	either a logical indicating whether label(s) should be plotted below the bar(s) on the right hand side, or a character string or vector giving the label(s) (see 'Details').
`interactive`	a logical indicating whether variables can be switched interactively (see 'Details').
`...`	further graphical parameters to be passed to `title` and `axis`.

Details

If more than one variable is supplied, the bars for the variable of interest are split according to missingness in the additional variables.

If `only.miss=TRUE`, the missing values in the variable of interest are visualized by one bar on the right hand side. If additional variables are supplied, this bar is again split into two parts according to missingness in the additional variables.

Otherwise, a small barplot consisting of two bars is drawn on the right hand side. The first bar corresponds to observed values in the variable of interest and the second bar to missing values. Since these two bars are not on the same scale as the main barplot, a second y-axis is plotted on the right (if `axes=TRUE`). Each of the two bars are again split into two parts according to missingness in the additional variables. Note that this display does not make sense if only one variable is supplied, therefore `only.miss` is ignored in that case.

If `interactive=TRUE`, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session. When switching to a continuous variable, a histogram is plotted rather than a barplot.

Value

a numeric vector giving the coordinates of the midpoints of the bars.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in VIM. For back compatibility, the arguments axisnames, names.arg and names.miss can still be supplied to ... and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use labels and miss.labels instead.

Author(s)

Andreas Alfons

See Also

spineMiss, histMiss

Examples

```
data(sleep, package = "VIM")
x <- sleep[, c("Exp", "Sleep")]
barMiss(x)
barMiss(x, only.miss = FALSE)
```

---

bgmap *Backgound map*

Description

Plot a background map.

Usage

```
bgmap(map, add=FALSE, ...)
```

Arguments

map	either a matrix or data.frame with two columns, a list with components x and y, or an object of any class that can be used for maps and provides its own plot method (e.g., "SpatialPolygons" from package sp). A list of the previously mentioned types can also be provided.
add	a logical indicating whether map should be added to an already existing plot (the default is FALSE).
...	further arguments and graphical parameters to be passed to plot and/or lines.

Author(s)

Andreas Alfons

See Also

`growdotMiss`, `mapMiss`

Examples

```
data(kola.background, package = "VIM")
bgmap(kola.background)
```

`chorizonDL` *C-horizon of the Kola data with missing values*

Description

This data set is the same as the `chorizon` data set in package `mvoutlier`, except that values below the detection limit are coded as `NA`.

Usage

```
data(chorizonDL)
```

Format

A data frame with 606 observations on the following 110 variables.

`*ID` a numeric vector

`XCOO` a numeric vector

`YCOO` a numeric vector

`Ag` a numeric vector

`Ag_INAA` a numeric vector

`Al` a numeric vector

`Al2O3` a numeric vector

`As` a numeric vector

`As_INAA` a numeric vector

`Au_INAA` a numeric vector

`B` a numeric vector

`Ba` a numeric vector

`Ba_INAA` a numeric vector

`Be` a numeric vector

`Bi` a numeric vector

`Br_IC` a numeric vector
`Br_INAA` a numeric vector
`Ca` a numeric vector
`Ca_INAA` a numeric vector
`CaO` a numeric vector
`Cd` a numeric vector
`Ce_INAA` a numeric vector
`Cl_IC` a numeric vector
`Co` a numeric vector
`Co_INAA` a numeric vector
`EC` a numeric vector
`Cr` a numeric vector
`Cr_INAA` a numeric vector
`Cs_INAA` a numeric vector
`Cu` a numeric vector
`Eu_INAA` a numeric vector
`F_IC` a numeric vector
`Fe` a numeric vector
`Fe_INAA` a numeric vector
`Fe2O3` a numeric vector
`Hf_INAA` a numeric vector
`Hg` a numeric vector
`Hg_INAA` a numeric vector
`Ir_INAA` a numeric vector
`K` a numeric vector
`K2O` a numeric vector
`La` a numeric vector
`La_INAA` a numeric vector
`Li` a numeric vector
`LOI` a numeric vector
`Lu_INAA` a numeric vector
`wt_INAA` a numeric vector
`Mg` a numeric vector
`MgO` a numeric vector
`Mn` a numeric vector
`MnO` a numeric vector
`Mo` a numeric vector

`Mo_INAA` a numeric vector
`Na` a numeric vector
`Na_INAA` a numeric vector
`Na2O` a numeric vector
`Nd_INAA` a numeric vector
`Ni` a numeric vector
`Ni_INAA` a numeric vector
`NO3_IC` a numeric vector
`P` a numeric vector
`P2O5` a numeric vector
`Pb` a numeric vector
`pH` a numeric vector
`PO4_IC` a numeric vector
`Rb` a numeric vector
`S` a numeric vector
`Sb` a numeric vector
`Sb_INAA` a numeric vector
`Sc` a numeric vector
`Sc_INAA` a numeric vector
`Se` a numeric vector
`Se_INAA` a numeric vector
`Si` a numeric vector
`SiO2` a numeric vector
`Sm_INAA` a numeric vector
`Sn_INAA` a numeric vector
`SO4_IC` a numeric vector
`Sr` a numeric vector
`Sr_INAA` a numeric vector
`SUM_XRF` a numeric vector
`Ta_INAA` a numeric vector
`Tb_INAA` a numeric vector
`Te` a numeric vector
`Th` a numeric vector
`Th_INAA` a numeric vector
`Ti` a numeric vector
`TiO2` a numeric vector
`U_INAA` a numeric vector

`V` a numeric vector

`W_INAA` a numeric vector

`Y` a numeric vector

`Yb_INAA` a numeric vector

`Zn` a numeric vector

`Zn_INAA` a numeric vector

`ELEV` a numeric vector

`*COUN` a numeric vector

`*ASP` a numeric vector

`TOPC` a numeric vector

`LITO` a numeric vector

`Al_XRF` a numeric vector

`Ca_XRF` a numeric vector

`Fe_XRF` a numeric vector

`K_XRF` a numeric vector

`Mg_XRF` a numeric vector

`Mn_XRF` a numeric vector

`Na_XRF` a numeric vector

`P_XRF` a numeric vector

`Si_XRF` a numeric vector

`Ti_XRF` a numeric vector

Note

For a more detailed description of this data set, see `chorizon` in package `mvoutlier`.

Source

Kola Project (1993-1998)

References

Reimann, C., Filzmoser, P., Garrett, R.G. and Dutter, R. (2008) *Statistical Data Analysis Explained: Applied Environmental Statistics with R*. Wiley.

See Also

`chorizon`

Examples

```
data(chorizonDL, package = "VIM")
summary(chorizonDL)
```

`colormapMiss` *Colored map with information about missing values*

Description

Colored map in which the proportion or amount of missing values in each region is coded according to a continuous or discrete color scheme. The sequential color palette may thereby be computed in the *HCL* or the *RGB* color space.

Usage

```
colormapMiss(x, region, map, prop = TRUE,
             polysRegion = 1:length(x), range = NULL, n = NULL,
             col = "red", gamma = 2.2, fixup = TRUE,
             coords = NULL, numbers = TRUE, digits = 2,
             cex.numbers = 0.8, col.numbers = par("fg"),
             legend = TRUE, interactive = TRUE, ...)

colormapMissLegend(xleft, ybottom, xright, ytop, cmap,
                   n = 1000, horizontal = TRUE, digits = 2,
                   cex.numbers = 0.8, col.numbers = par("fg"),
                   ...)
```

Arguments

`x`	a numeric vector.
`region`	a vector or factor of the same length as `x` giving the regions.
`map`	an object of any class that contains polygons and provides its own plot method (e.g., `"SpatialPolygons"` from package `sp`).
`prop`	a logical indicating whether the proportion of missing values should be used rather than the total amount.
`polysRegion`	a numeric vector specifying the region that each polygon belongs to.
`range`	a numeric vector of length two specifying the range (minimum and maximum) of the proportion or amount of missing values to be used for the color scheme.
`n`	for `colormapMiss`, the number of equally spaced cut-off points for a discretized color scheme. If this is not a positive integer, a continuous color scheme is used (the default). In the latter case, the number of rectangles to be drawn in the legend can be specified in `colormapMissLegend`. A reasonably large number makes it appear continuously.
`col`	the color range (start end end) to be used. RGB colors may be specified as character strings or as objects of class "`RGB`". HCL colors need to be specified as objects of class "`polarLUV`". If only one color is supplied, it is used as end color, while the start color is taken to be transparent for RGB or white for HCL.
`gamma`	numeric; the display *gamma* value (see `hex`).

fixup	a logical indicating whether the colors should be corrected to valid RGB values (see hex).
coords	a matrix or data.frame with two columns giving the coordinates for the labels.
numbers	a logical indicating whether the corresponding proportions or numbers of missing values should be used as labels for the regions.
digits	the number of digits to be used in the labels (in case of proportions).
cex.numbers	the character expansion factor to be used for the labels.
col.numbers	the color to be used for the labels.
legend	a logical indicating whether a legend should be plotted.
interactive	a logical indicating whether more detailed information about missing values should be displayed interactively (see 'Details').
xleft	left *x* position of the legend.
ybottom	bottom *y* position of the legend.
xright	right *x* position of the legend.
ytop	top *y* position of the legend.
cmap	a list as returned by colormapMiss that contains the required information for the legend.
horizontal	a logical indicating whether the legend should be drawn horizontally or vertically.
...	further arguments to be passed to plot.

Details

The proportion or amount of missing values in x of each region is coded according to a continuous or discrete color scheme in the color range defined by col. In addition, the proportions or numbers can be shown as labels in the regions.

If interactive is TRUE, clicking in a region displays more detailed information about missing values on the R console. Clicking outside the borders quits the interactive session.

Value

colormapMiss returns a list with the following components:

nmiss	a numeric vector containing the number of missing values in each region.
nobs	a numeric vector containing the number of observations in each region.
pmiss	a numeric vector containing the proportion of missing values in each region.
prop	a logical indicating whether the proportion of missing values have been used rather than the total amount.
range	the range of the proportion or amount of missing values corresponding to the color range.
n	either a positive integer giving the number of equally spaced cut-off points for a discretized color scheme, or NULL for a continuous color scheme.

`start`	the start color of the color scheme.
`end`	the end color of the color scheme.
`space`	a character string giving the color space (either `"rgb"` for RGB colors or `"hcl"` for HCL colors).
`gamma`	numeric; the display *gamma* value (see `hex`).
`fixup`	a logical indicating whether the colors have been corrected to valid RGB values (see `hex`).

Note

Some of the argument names and positions have changed with versions 1.3 and 1.4 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `cex.text` and `col.text` can still be supplied to `...` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `cex.numbers` and `col.numbers` instead.

Author(s)

Andreas Alfons

See Also

`colSequence`, `growdotMiss`, `mapMiss`

`colSequence` *HCL and RGB color sequences*

Description

Compute color sequences by linear interpolation based on a continuous color scheme between certain start and end colors. Color sequences may thereby be computed in the *HCL* or *RGB* color space.

Usage

```
colSequence(p, start, end, space = c("hcl", "rgb"), ...)

colSequenceHCL(p, start, end, gamma = 2.2, fixup = TRUE, ...)

colSequenceRGB(p, start, end, gamma = 2.2, fixup = TRUE, ...)
```

Arguments

p	a numeric vector in $[0, 1]$ giving values to be used for interpolation between the start and end color (0 corresponds to the start color, 1 to the end color).
start, end	the start and end color, respectively. For HCL colors, each can be supplied as a vector of length three (hue, chroma, luminance) or an object of class "polarLUV". For RGB colors, each can be supplied as a character string, a vector of length three (red, green, blue) or an object of class "RGB".
space	character string; if start and end are both numeric, this determines whether they refer to HCL or RGB values. Possible values are "hcl" (for the HCL space) or "rgb" (for the RGB space).
gamma	numeric; the display *gamma* value (see hex).
fixup	a logical indicating whether the colors should be corrected to valid RGB values (see hex).
...	for colSequence, additional arguments to be passed to colSequenceHCL or colSequenceRGB. For colSequenceHCL and colSequenceRGB, additional arguments to be passed to hex.

Value

A character vector containing hexadecimal strings of the form "#RRGGBB".

Author(s)

Andreas Alfons

References

Zeileis, A., Hornik, K., Murrell, P. (2009) Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, **53 (9)**, 1259–1270.

See Also

hex, sequential_hcl

Examples

```
p <- c(0, 0.3, 0.55, 0.8, 1)

## HCL colors
colSequence(p, c(0, 0, 100), c(0, 100, 50))
colSequence(p, polarLUV(L=90, C=30, H=90), c(0, 100, 50))

## RGB colors
colSequence(p, c(1, 1, 1), c(1, 0, 0), space="rgb")
colSequence(p, RGB(1, 1, 0), "red")
```

`count` *Count number of infinite or missing values*

Description

Count the number of infinite or missing values in a vector.

Usage

```
countInf(x)

countNA(x)
```

Arguments

`x`	a vector.

Value

`countInf` returns the number of infinite values in `x`.

`countNA` returns the number of missing values in `x`.

Author(s)

Andreas Alfons

Examples

```
data(sleep, package="VIM")
countInf(log(sleep$Dream))
countNA(sleep$Dream)
```

`growdotMiss` *Growing dot map with information about missing values*

Description

Map with dots whose sizes correspond to the values in a certain variable. Observations with missing values in additional variables are highlighted.

Usage

```
growdotMiss(x, coords, map, pos=1, selection = c("any","all"),
            log = FALSE, col = c("skyblue","red","skyblue4","red4"),
            border = par("bg"), alpha = NULL, scale = NULL,
            size = NULL, exp = c(0, 0.95, 0.05),
            col.map = grey(0.5), legend = TRUE,
            legtitle = "Legend", cex.legtitle = par("cex"),
            cex.legtext = par("cex"), ncircles = 6, ndigits = 1,
            interactive = TRUE, ...)

bubbleMiss(...)
```

Arguments

`x`	a vector, matrix or `data.frame`.
`coords`	a matrix or `data.frame` with two columns giving the spatial coordinates of the observations.
`map`	a background map to be passed to `bgmap`.
`pos`	a numeric value giving the index of the variable determining the dot sizes.
`selection`	the selection method for highlighting missing values in multiple additional variables. Possible values are `"any"` (highlighting of missing values in *any* of the additional variables) and `"all"` (highlighting of missing values in *all* of the additional variables).
`log`	a logical indicating whether the variable given by `pos` should be log-transformed.
`col`	a vector of length four giving the colors to be used in the plot. If only one color is supplied, it is used for the borders of non-highlighted dots and the surface area of highlighted dots. Else if two colors are supplied, they are recycled.
`border`	a vector of length four giving the colors to be used for the borders of the growing dots. Use `NA` to omit borders.
`alpha`	a numeric value between 0 and 1 giving the level of transparency of the colors, or `NULL`. This can be used to prevent overplotting.
`scale`	scaling factor of the map.
`size`	a vector of length two giving the sizes for the smallest and largest dots.
`exp`	a vector of length three giving the factors that define the shape of the exponential function (see 'Details').
`col.map`	the color to be used for the background map.
`legend`	a logical indicating whether a legend should be plotted.
`legtitle`	the title for the legend.
`cex.legtitle`	the character expansion factor to be used for the title of the legend.
`cex.legtext`	the character expansion factor to be used in the legend.
`ncircles`	the number of circles displayed in the legend.
`ndigits`	the number of digits displayed in the legend. Note that \ this is just a suggestion (see `format`).

`interactive`	a logical indicating whether information about certain observations can be displayed interactively (see 'Details').
`...`	for `growdotMiss`, further arguments and graphical parameters to be passed to `bgmap`. For `bubbleMiss`, the arguments to be passed to `growdotMiss`.

Details

The smallest dots correspond to the 10% quantile and the largest dots to the 99% quantile. In between, the dots grow exponentially, with `exp` defining the shape of the exponential function. Missings in the variable of interest will be drawn as rectangles.

If `interactive=TRUE`, detailed information for an observation can be printed on the console by clicking on the corresponding point. Clicking in a region that does not contain any points quits the interactive session.

Note

The function was renamed to `growdotMiss` in version 1.3. `bubbleMiss` is a (deprecated) wrapper for `growdotMiss` for back compatibility with older versions. However, due to extended functionality, some of the argument positions have changed.

The code is based on `bubbleFIN` from package `StatDA`.

Author(s)

Andreas Alfons

See Also

`bgmap`, `mapMiss`, `colormapMiss`

Examples

```
data(chorizonDL, package = "VIM")
data(kola.background, package = "VIM")
x <- chorizonDL[, c("Ca","As", "Bi")]
coo <- chorizonDL[, c("XCOO", "YCOO")]
growdotMiss(x, coo, kola.background, border = "white")
```

`histMiss` *Histogram with information about missing values*

Description

Histogram with highlighting of missing values in other variables by splitting each bin into two parts. Additionally, information about missing values in the variable of interest is shown on the right hand side.

Usage

```
histMiss(x, pos = 1, selection = c("any","all"),
         breaks = "Sturges", right = TRUE,
         col = c("skyblue","red","skyblue4","red4"),
         border = NULL, main = NULL, sub = NULL,
         xlab = NULL, ylab = NULL, axes = TRUE,
         only.miss = TRUE, miss.labels = axes,
         interactive = TRUE, ...)
```

Arguments

x	a vector, matrix or data.frame.
pos	a numeric value giving the index of the variable of interest. Additional variables in x are used for highlighting.
selection	the selection method for highlighting missing values in multiple additional variables. Possible values are "any" (highlighting of missing values in *any* of the additional variables) and "all" (highlighting of missing values in *all* of the additional variables).
breaks	either a character string naming an algorithm to compute the breakpoints (see hist), or a numeric value giving the number of cells.
right	logical; if TRUE, the histogram cells are right-closed (left-open) intervals.
col	a vector of length four giving the colors to be used. If only one color is supplied, the bars are transparent and the supplied color is used for highlighting. Else if two colors are supplied, they are recycled.
border	the color to be used for the border of the cells. Use border=NA to omit borders.
main, sub	main and sub title.
xlab, ylab	axis labels.
axes	a logical indicating whether axes should be drawn on the plot.
only.miss	logical; if TRUE, the missing values in the first variable are visualized by a single bar. Otherwise, a small barplot is drawn on the right hand side (see 'Details').
miss.labels	either a logical indicating whether label(s) should be plotted below the bar(s) on the right hand side, or a character string or vector giving the label(s) (see 'Details').
interactive	a logical indicating whether the variables can be switched interactively (see 'Details').
...	further graphical parameters to be passed to title and axis.

Details

If more than one variable is supplied, the bins for the variable of interest will be split according to missingness in the additional variables.

If only.miss=TRUE, the missing values in the variable of interest are visualized by one bar on the right hand side. If additional variables are supplied, this bar is again split into two parts according to missingness in the additional variables.

Otherwise, a small barplot consisting of two bars is drawn on the right hand side. The first bar corresponds to observed values in the variable of interest and the second bar to missing values. Since these two bars are not on the same scale as the main barplot, a second y-axis is plotted on the right (if `axes=TRUE`). Each of the two bars are again split into two parts according to missingness in the additional variables. Note that this display does not make sense if only one variable is supplied, therefore `only.miss` is ignored in that case.

If `interactive=TRUE`, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session. When switching to a categorical variable, a barplot is produced rather than a histogram.

Value

a list with the following components:

`breaks`	the breakpoints.
`counts`	the number of observations in each cell.
`missings`	the number of highlighted observations in each cell.
`mids`	the cell midpoints.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `axisnames` and `names.miss` can still be supplied to `...` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `miss.labels` instead.

Author(s)

Andreas Alfons, Matthias Templ

See Also

`spineMiss`, `barMiss`

Examples

```
data(tao, package = "VIM")
x <- tao[, c("Air.Temp", "Humidity")]
histMiss(x)
histMiss(x, only.miss = FALSE)
```

`hotdeck` *Hot-Deck Imputation*

Description

Implementation of the popular Sequential, Random (within a domain) hot-deck algorithm for imputation.

Usage

```
hotdeck(data, variable=colnames(data), ord_var=NULL,domain_var=NULL,
    makeNA=NULL,NAcond=NULL,impNA=TRUE,donorcond=NULL,
    imp_var=TRUE,imp_suffix="imp")
```

Arguments

`data`	data.frame or matrix
`variable`	variables where missing values should be imputed
`ord_var`	variables for sorting the data set before imputation
`domain_var`	variables for building domains and impute within these domains
`makeNA`	vector of values, that should be converted to NA
`NAcond`	a condition for imputing a NA
`impNA`	TRUE/FALSE whether NA should be imputed
`donorcond`	condition for the donors e.g. ">5"
`imp_var`	TRUE/FALSE if a TRUE/FALSE variables for each imputed variable should be created show the imputation status
`imp_suffix`	suffix for the TRUE/FALSE variables showing the imputation status

Value

the imputed data set.

Author(s)

Alexander Kowarik

Examples

```
data(sleep)
sleepI <- hotdeck(sleep)
sleepI2 <- hotdeck(sleep,ord_var="BodyWgt",domain_var="Pred")
```

`initialise` *Initialization of missing values*

Description

Rough estimation of missing values in a vector according to its type.

Usage

```
initialise(x)
```

Arguments

`x`	a vector.

Details

Missing values are imputed with the mean for vectors of class `"numeric"`, with the median for vectors of class `"integer"`, and with the mode for vectors of class `"factor"`. Hence, `x` should be prepared in the following way: assign class `"numeric"` to numeric vectors, assign class `"integer"` to ordinal vectors, and assign class `"factor"` to nominal or binary vectors.

Value

the initialized vector.

Note

The function is used internally by some imputation algorithms.

Author(s)

Matthias Templ, modifications by Andreas Alfons

`irmi` *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

```
irmi(x, eps=5, maxit=100, mixed=NULL, count=NULL, step=FALSE,
    robust=FALSE, takeAll=TRUE,
    noise=TRUE, noise.factor=1, force=FALSE,
    robMethod="MM", force.mixed=TRUE, mi=1,
    addMixedFactors=FALSE, 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
count	column index of count 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.
addMixedFactors	if factor variables for the mixed variables should be created for the regression models
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)
```

`kNN` *k-Nearest Neighbour Imputation*

Description

k-Nearest Neighbour Imputation based on a variation of the Gower Distance for numerical, categorical, ordered and semi-continous variables.

Usage

```
kNN(data, variable=colnames(data), metric=NULL, k=5, dist_var=colnames(data),weight
        numFun = median, catFun=maxCat,
        makeNA=NULL,NAcond=NULL, impNA=TRUE, donorcond=NULL,mixed=vector(),trace=FA
        imp_var=TRUE,imp_suffix="imp",addRandom=FALSE)
sampleCat(x)
maxCat(x)
gowerD(data.x, data.y = data.x, weights=NULL,numerical,factors,orders,mixed,levOrde
which.minN(x,n)
```

Arguments

`data`	data.frame or matrix
`variable`	variables where missing values should be imputed
`metric`	metric to be used for calculating the distances between
`k`	number of Nearest Neighbours used
`dist_var`	names or variables to be used for distance calculation
`weights`	weights for the variables for distance calculation
`numFun`	function for aggregating the k Nearest Neighbours in the case of a numerical variable
`catFun`	function for aggregating the k Nearest Neighbours in the case of a categorical variable
`makeNA`	vector of values, that should be converted to NA

`NAcond`	a condition for imputing a NA
`impNA`	TRUE/FALSE whether NA should be imputed
`donorcond`	condition for the donors e.g. ">5"
`trace`	TRUE/FALSE if additional information about the imputation process should be printed
`imp_var`	TRUE/FALSE if a TRUE/FALSE variables for each imputed variable should be created show the imputation status
`imp_suffix`	suffix for the TRUE/FALSE variables showing the imputation status
`addRandom`	TRUE/FALSE if an additional random variable should be added for distance calculation
`x`	factor or character vector / numerical vector for which.minN
`data.x`	data frame or matrix
`data.y`	data frame or matrix
`numerical`	names of numerical variables
`factors`	names of factors
`orders`	names of ordered variables
`mixed`	names of mixed variables
`levOrders`	list of the ordered levels for each factor
`n`	number of ordered smallest values

Details

The function sampleCat samples with probabilites corresponding to the occurrence of the level in the NNs. The function maxCat chooses the level with the most occurrences and random if the maximum is not unique. The function gowerD is used by kNN to compute the distances for numerical, factor ordered and semi-continous variables. The function which.minN is used by kNN.

Value

the imputed data set.

Author(s)

Alexander Kowarik

Examples

```
data(sleep)
kNN(sleep)
```

`kola.background` *Background map for the Kola project data*

Description

Coordinates of the Kola background map.

Usage

```
data(kola.background)
```

Source

Kola Project (1993-1998)

References

Reimann, C., Filzmoser, P., Garrett, R.G. and Dutter, R. (2008) *Statistical Data Analysis Explained: Applied Environmental Statistics with R*. Wiley, 2008.

Examples

```
data(kola.background, package = "VIM")
bgmap(kola.background)
```

`mapMiss` *Map with information about missing values*

Description

Map of observed and missing values.

Usage

```
mapMiss(x, coords, map, selection = c("any","all"),
        col = c("skyblue","red"), alpha = NULL,
        pch = c(19,15), col.map = grey(0.5), legend = TRUE,
        interactive = TRUE, ...)
```

Arguments

x	a vector, matrix or data.frame.
coords	a data.frame or matrix with two columns giving the spatial coordinates of the observations.
map	a background map to be passed to bgmap.
selection	the selection method for displaying missing values in the map. Possible values are "any" (display missing values in *any* variable) and "all" (display missing values in *all* variables).
col	a vector of length two giving the colors to be used for observed and missing values. If a single color is supplied, it is used for both.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
pch	a vector of length two giving the plot characters to be used for observed and missing values. If a single plot character is supplied, it will be used for both.
col.map	the color to be used for the background map.
legend	a logical indicating whether a legend should be plotted.
interactive	a logical indicating whether information about selected observations can be displayed interactively (see 'Details').
...	further graphical parameters to be passed to bgmap and points.

Details

If interactive=TRUE, detailed information for an observation can be printed on the console by clicking on the corresponding point. Clicking in a region that does not contain any points quits the interactive session.

Author(s)

Matthias Templ, Andreas Alfons

See Also

bgmap, bubbleMiss, colormapMiss

Examples

```
data(chorizonDL, package = "VIM")
data(kola.background, package = "VIM")
x <- chorizonDL[, c("As", "Bi")]
coo <- chorizonDL[, c("XCOO", "YCOO")]
mapMiss(x, coo, kola.background)
```

`marginmatrix` *Marginplot Matrix*

Description

Create a scatterplot matrix with information about missing values in the plot margins of each panel.

Usage

```
marginmatrix(x, col = c("skyblue","red","red4"),
             alpha = NULL, ...)

TKRmarginmatrix(x, col = c("skyblue","red","red4"),
                alpha = NULL, ..., hscale = NULL,
                vscale = NULL, TKRpar = list())
```

Arguments

`x`	a matrix or `data.frame`.
`col`	a vector of length three giving the colors to be used in the marginplots in the off-diagonal panels. The first color is used for the scatterplot and the boxplots for the available data, the second color for the univariate scatterplots and boxplots for the missing values in one variable, and the third color for the frequency of missing values in both variables (see 'Details'). If only one color is supplied, it is used for the bivariate and univariate scatterplots and the boxplots for missing values in one variable, whereas the boxplots for the available data are transparent. Else if two colors are supplied, the second one is recycled.
`alpha`	a numeric value between 0 and 1 giving the level of transparency of the colors, or `NULL`. This can be used to prevent overplotting.
`...`	further arguments and graphical parameters to be passed to `pairsVIM` and `marginplot`. `par("oma")` will be set appropriately unless supplied (see `par`).
`hscale`	horizontal scale factor for plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of variables.
`vscale`	vertical scale factor for the plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of variables.
`TKRpar`	a list of graphical parameters to be set for the plot to be embedded in a *Tcl/Tk* window (see 'Details' and `par`).

Details

`marginmatrix` uses `pairsVIM` with a panel function based on `marginplot`.

The graphical parameter `oma` will be set unless supplied as an argument.

`TKRmarginmatrix` behaves like `marginmatrix`, but uses `tkrplot` to embed the plot in a *Tcl/Tk* window. This is useful if the number of variables is large, because scrollbars allow to move from one part of the plot to another.

Author(s)

Andreas Alfons

See Also

marginplot, pairsVIM, scattmatrixMiss

Examples

```
data(sleep, package = "VIM")
x <- sleep[, 1:5]
x[,c(1,2,4)] <- log10(x[,c(1,2,4)])
marginmatrix(x)
```

---

marginplot *Scatterplot with additional information in the margins*

Description

In addition to a standard scatterplot, information about missing values is shown in the plot margins.

Usage

```
marginplot(x, col = c("skyblue","red","red4"),
           alpha = NULL, pch = c(1,16), cex = par("cex"),
           numbers = TRUE, cex.numbers = par("cex"),
           zeros = FALSE, xlim = NULL, ylim = NULL,
           main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
           ann = par("ann"), axes = TRUE, frame.plot = axes, ...)
```

Arguments

x	a matrix or data.frame with two columns.
col	a vector of length three giving the colors to be used in the plot. The first color is used for the scatterplot and the boxplots for the available data, the second color for the univariate scatterplots and boxplots for the missing values in one variable, and the third color for the frequency of missing values in both variables (see 'Details'). If only one color is supplied, it is used for the bivariate and univariate scatterplots and the boxplots for missing values in one variable, whereas the boxplots for the available data are transparent. Else if two colors are supplied, the second one is recycled.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
pch	a vector of length two giving the plot symbols to be used for the scatterplot and the univariate scatterplots. If a single plot character is supplied, it is used for the scatterplot and the default value will be used for the univariate scatterplots (see 'Details').

`cex`	the character expansion factor to be used for the bivariate and univariate scatterplots.
`numbers`	a logical indicating whether the frequencies of missing values should be displayed in the lower left of the plot (see 'Details').
`cex.numbers`	the character expansion factor to be used for the frequencies of the missing values.
`zeros`	a logical vector of length two indicating whether the variables are semi-continuous, i.e., contain a considerable amount of zeros. If `TRUE`, only the non-zero observations are used for drawing the respective boxplot. If a single logical is supplied, it is recycled.
`xlim, ylim`	axis limits.
`main, sub`	main and sub title.
`xlab, ylab`	axis labels.
`ann`	a logical indicating whether plot annotation (`main`, `sub`, `xlab`, `ylab`) should be displayed.
`axes`	a logical indicating whether both axes should be drawn on the plot. Use graphical parameter `"xaxt"` or `"yaxt"` to suppress only one of the axes.
`frame.plot`	a logical indicating whether a box should be drawn around the plot.
`...`	further graphical parameters to be passed down (see `par`).

Details

Boxplots for available and missing data, as well as univariate scatterplots for missing values in one variable are shown in the plot margins.

Furthermore, the frequencies of the missing values can be displayed by a number (lower left of the plot). The number in the lower left corner is the number of observations that are missing in both variables.

Note

Some of the argument names and positions have changed with versions 1.3 and 1.4 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the argument `cex.text` can still be supplied to `...` and is handled correctly. Nevertheless, it is deprecated and no longer documented. Use `cex.numbers` instead.

Author(s)

Andreas Alfons, Matthias Templ

See Also

`scattMiss`

Examples

```
data(tao, package = "VIM")
marginplot(tao[,c("Air.Temp", "Humidity")])
data(chorizonDL, package = "VIM")
marginplot(log10(chorizonDL[,c("CaO", "Bi")]))
```

`matrixplot` *Matrix plot*

Description

Create a matrix plot, in which all cells of a data matrix are visualized by rectangles. Available data is coded according to a continuous color scheme, while missing data is visualized by a clearly distinguishable color.

Usage

```
matrixplot(x, sortby = NULL, col = "red", gamma = 2.2,
           fixup = TRUE, xlim = NULL, ylim = NULL, main = NULL,
           sub = NULL, xlab = NULL, ylab = NULL,
           axes = TRUE, labels = axes, xpd = NULL,
           interactive = TRUE, ...)

TKRmatrixplot(x, ..., hscale = NULL,
              vscale = NULL, TKRpar = list())

iimagMiss(x, sortby = NULL, col = "red", main = NULL,
         sub = NULL, xlab = NULL, ylab = NULL,
         xlim = NULL, ylim = NULL, axes = TRUE,
         xaxlabels = NULL, las = 3, interactive = TRUE,
         ...)
```

Arguments

`x`	a matrix or `data.frame`.
`sortby`	a numeric or character value specifying the variable to sort the data matrix by, or `NULL` to plot without sorting.
`col`	the colors to be used in the plot. RGB colors may be specified as character strings or as objects of class "`RGB`". HCL colors need to be specified as objects of class "`polarLUV`". If only one color is supplied, it is used for missing data and a greyscale is used for available data. If two colors are supplied, the first is used as end color for the available data, while the start color is taken to be transparent for RGB or white for HCL. Missing data is visualized by the second color in this case. If three colors are supplied, the first is used as start color and the second as end color for the available data, while the third color is used for missing data.

`gamma`	numeric; the display *gamma* value (see `hex`).
`fixup`	a logical indicating whether the colors should be corrected to valid RGB values (see `hex`).
`xlim, ylim`	axis limits.
`main, sub`	main and sub title.
`xlab, ylab`	axis labels.
`axes`	a logical indicating whether axes should be drawn on the plot.
`labels`	either a logical indicating whether labels should be plotted below each column, or a character vector giving the labels.
`xpd`	a logical indicating whether the rectangles should be allowed to go outside the plot region. If `NULL`, it defaults to `TRUE` unless axis limits are specified.
`interactive`	a logical indicating whether a variable to be used for sorting can be selected interactively (see 'Details').
`xaxlabels`	a character vector containing the labels for the columns. If `NULL`, the column names of `x` will be used.
`las`	the style of axis labels (see `par`).
`...`	for `matrixplot` and `iimagMiss`, further graphical parameters to be passed to `plot.window`, `title` and `axis`. For `TKRmatrixplot`, further arguments to be passed to `matrixplot`.
`hscale`	horizontal scale factor for plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of variables.
`vscale`	vertical scale factor for the plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of observations.
`TKRpar`	a list of graphical parameters to be set for the plot to be embedded in a *Tcl/Tk* window (see 'Details' and `par`).

Details

In a *matrix plot*, all cells of a data matrix are visualized by rectangles. Available data is coded according to a continuous color scheme. To compute the colors via interpolation, the variables are first scaled to the interval

$$[0, 1]$$

. Missing values can then be visualized by a clearly distinguishable color. It is thereby possible to use colors in the *HCL* or *RGB* color space. A simple way of visualizing the magnitude of the available data is to apply a greyscale, which has the advantage that missing values can easily be distinguished by using a color such as red. Note that `-Inf` and `Inf` are always assigned the begin and end color, respectively, of the continuous color scheme.

Additionally, the observations can be sorted by the magnitude of a selected variable. If `interactive` is `TRUE`, clicking in a column redraws the plot with observations sorted by the corresponding variable. Clicking anywhere outside the plot region quits the interactive session.

`TKRmatrixplot` behaves like `matrixplot`, but uses `tkrplot` to embed the plot in a *Tcl/Tk* window. This is useful if the number of observations and/or variables is large, because scrollbars allow to move from one part of the plot to another.

Note

This is a much more powerful extension to the function `imagmiss` in the former CRAN package `dprep`.

`iimagMiss` is deprecated and may be omitted in future versions of `VIM`. Use `matrixplot` instead.

Author(s)

Andreas Alfons, Matthias Templ

Examples

```
data(sleep, package = "VIM")
x <- sleep[, -(8:10)]
x[,c(1,2,4,6,7)] <- log10(x[,c(1,2,4,6,7)])
matrixplot(x, sortby = "BrainWgt")
```

---

mosaicMiss *Mosaic plot with information about missing values*

Description

Create a mosaic plot with information about missing values.

Usage

```
mosaicMiss(x, highlight = NULL, selection = c("any", "all"),
           plotvars = NULL, col = c("skyblue", "red"),
           labels = NULL, miss.labels = TRUE, ...)
```

Arguments

`x`	a matrix or `data.frame`.
`highlight`	a vector giving the variables to be used for highlighting. If `NULL` (the default), all variables are used for highlighting.
`selection`	the selection method for highlighting missing values in multiple highlight variables. Possible values are `"any"` (highlighting of missing values in *any* of the highlight variables) and `"all"` (highlighting of missing values in *all* of the highlight variables).
`plotvars`	a vector giving the categorical variables to be plotted. If `NULL` (the default), all variables are plotted.
`col`	a vector of length two giving the colors to be used for observed and missing data. If only one color is supplied, the tiles corresponding to observed data are transparent and the supplied color is used for highlighting.
`labels`	a list of arguments for the labeling function `labeling_border`.

`miss.labels`	either a logical indicating whether labels should be plotted for observed and missing (highlighted) data, or a character vector giving the labels.
`...`	additional arguments to be passed to `mosaic`.

Details

Mosaic plots are graphical representations of multi-way contingency tables. The frequencies of the different cells are visualized by area-proportional rectangles (tiles). Additional tiles are be used to display the frequencies of missing values. Furthermore, missing values in a certain variable or combination of variables can be highlighted in order to explore their structure.

Value

An object of class `"structable"` is returned invisibly.

Note

This function uses the highly flexible `strucplot` framework of package `vcd`.

Author(s)

Andreas Alfons

References

Meyer, D., Zeileis, A. and Hornik, K. (2006) The `strucplot` framework: Visualizing multi-way contingency tables with **vcd**. *Journal of Statistical Software*, **17 (3)**, 1–48.

See Also

`spineMiss`, `mosaic`

Examples

```
data(sleep, package = "VIM")
mosaicMiss(sleep, highlight = 4,
    plotvars = 8:10, miss.labels = FALSE)
```

`pairsVIM` *Scatterplot Matrices*

Description

Create a scatterplot matrix.

Usage

```
pairsVIM(x, ..., main = NULL, sub = NULL, panel = points,
         lower = panel, upper = panel, diagonal = NULL,
         labels = TRUE, pos.labels = NULL, cex.labels = NULL,
         font.labels = par("font"), layout = c("matrix","graph"),
         gap = 1)
```

Arguments

x	a matrix or data.frame.
main, sub	main and sub title.
panel	a function(x, y, ...), which is used to plot the contents of each off-diagonal panel of the display.
...	further arguments and graphical parameters to be passed down. par("oma") will be set appropriately unless supplied (see par).
lower, upper	separate panel functions to be used below and above the diagonal, respectively.
diagonal	optional function(x, ...) to be applied on the diagonal panels.
labels	either a logical indicating whether labels should be plotted in the diagonal panels, or a character vector giving the labels.
pos.labels	the vertical position of the labels in the diagonal panels.
cex.labels	the character expansion factor to be used for the labels.
font.labels	the font to be used for the labels.
layout	a character string giving the layout of the scatterplot matrix. Possible values are "matrix" (a matrix-like layout with the first row on top) and "graph" (a graph-like layout with the first row at the bottom).
gap	a numeric value giving the distance between the panels in margin lines.

Details

This function is the workhorse for marginmatrix and scattmatrixMiss.

The graphical parameter oma will be set unless supplied as an argument.

A panel function should not attempt to start a new plot, since the coordinate system for each panel is set up by pairsVIM.

Note

The code is based on pairs. Starting with version 1.4, infinite values are no longer removed before passing the x and y vectors to the panel functions.

Author(s)

Andreas Alfons

See Also

marginmatrix, scattmatrixMiss

Examples

```
data(sleep, package = "VIM")
x <- sleep[, -(8:10)]
x[,c(1,2,4,6,7)] <- log10(x[,c(1,2,4,6,7)])
pairsVIM(x)
```

`parcoordMiss` *Parallel coordinate plot with information about missing values*

Description

Parallel coordinate plot with adjustments for missing values. Missing values in the plotted variables may be represented by a point above the corresponding coordinate axis to prevent disconnected lines. In addition, observations with missing values in selected variables may be highlighted.

Usage

```
parcoordMiss(x, highlight = NULL, selection = c("any","all"),
             plotvars = NULL, plotNA = TRUE,
             col = c("skyblue","red","skyblue4","red4"),
             alpha = NULL, lty = par("lty"), xlim = NULL,
             ylim = NULL, main = NULL, sub = NULL,
             xlab = NULL, ylab = NULL, labels = TRUE,
             xpd = NULL, interactive = TRUE, ...)

TKRparcoordMiss(x, highlight = NULL, selection = c("any","all"),
                plotvars = NULL, plotNA = TRUE,
                col = c("skyblue","red","skyblue4","red4"),
                alpha = NULL, ..., hscale = NULL, vscale = 1,
                TKRpar = list())
```

Arguments

`x`	a matrix or `data.frame`.
`highlight`	a vector giving the variables to be used for highlighting. If `NULL` (the default), all variables are used for highlighting.
`selection`	the selection method for highlighting missing values in multiple highlight variables. Possible values are `"any"` (highlighting of missing values in *any* of the highlight variables) and `"all"` (highlighting of missing values in *all* of the highlight variables).
`plotvars`	a vector giving the variables to be plotted. If `NULL` (the default), all variables are plotted.
`col`	if `plotNA` is `TRUE`, a vector of length four giving the colors to be used for observations with different combinations of observed and missing values in the plot variables and highlight variables (vectors of length one or two are recycled). Otherwise, a vector of length two giving the colors for non-highlighted and highlighted observations (if a single color is supplied, it is used for both).

plotNA	a logical indicating whether missing values in the plot variables should be represented by a point above the corresponding coordinate axis to prevent disconnected lines.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
lty	if plotNA is TRUE, a vector of length four giving the line types to be used for observations with different combinations of observed and missing values in the plot variables and highlight variables (vectors of length one or two are recycled). Otherwise, a vector of length two giving the line types for non-highlighted and highlighted observations (if a single line type is supplied, it is used for both).
xlim, ylim	axis limits.
main, sub	main and sub title.
xlab, ylab	axis labels.
labels	either a logical indicating whether labels should be plotted below each coordinate axis, or a character vector giving the labels.
xpd	a logical indicating whether the lines should be allowed to go outside the plot region. If NULL, it defaults to TRUE unless axis limits are specified.
interactive	a logical indicating whether interactive features should be enabled (see 'Details').
...	for parcoordMiss, further graphical parameters to be passed down (see par). For TKRparcoordMiss, further arguments to be passed to parcoordMiss.
hscale	horizontal scale factor for plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of variables.
vscale	vertical scale factor for the plot to be embedded in a *Tcl/Tk* window (see 'Details').
TKRpar	a list of graphical parameters to be set for the plot to be embedded in a *Tcl/Tk* window (see 'Details' and par).

Details

In parallel coordinate plots, the variables are represented by parallel axes. Each observation of the scaled data is shown as a line. Observations with missing values in selected variables may thereby be highlighted. However, plotting variables with missing values results in disconnected lines, making it impossible to trace the respective observations across the graph. As a remedy, missing values may be represented by a point above the corresponding coordinate axis, which is separated from the main plot by a small gap and a horizontal line, as determined by plotNA. Connected lines can then be drawn for all observations. Nevertheless, a caveat of this display is that it may draw attention away from the main relationships between the variables.

If interactive is TRUE, it is possible switch between this display and the standard display without the separate level for missing values by clicking in the top margin of the plot. In addition, the variables to be used for highlighting can be selected interactively. Observations with missing values in any or in all of the selected variables are highlighted (as determined by selection). A variable can be added to the selection by clicking on a coordinate axis. If a variable is already selected, clicking on its coordinate axis removes it from the selection. Clicking anywhere outside the plot region (except the top margin, if missing values exist) quits the interactive session.

`TKRparcoordMiss` behaves like `parcoordMiss`, but uses `tkrplot` to embed the plot in a *Tcl/Tk* window. This is useful if the number of variables is large, because scrollbars allow to move from one part of the plot to another.

Note

Some of the argument names and positions have changed with versions 1.3 and 1.4 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `colcomb` and `xaxlabels` can still be supplied to `...` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `highlight` and `labels` instead.

Author(s)

Andreas Alfons, Matthias Templ

References

Wegman, E. J. (1990) Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association* **85 (411)**, 664–675.

See Also

`pbox`

Examples

```
data(chorizonDL, package = "VIM")
parcoordMiss(chorizonDL[,c(15,101:110)],
    plotvars=2:11, interactive = FALSE)
legend("top", col = c("skyblue", "red"), lwd = c(1,1),
    legend = c("observed in Bi", "missing in Bi"))
```

---

pbox *Parallel boxplots with information about missing values*

Description

Boxplot of one variable of interest plus information about missing values in other variables.

Usage

```
pbox(x, pos = 1, selection = c("none","any","all"),
     col = c("skyblue","red","red4"), numbers = TRUE,
     cex.numbers = par("cex"), xlim = NULL, ylim = NULL,
     main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
     axes = TRUE, frame.plot = axes, labels = axes,
     interactive = TRUE, ...)
```

```
TKRpbox(x, pos = 1, ..., hscale = NULL,
        vscale = 1, TKRpar = list())
```

Arguments

x	a vector, matrix or data.frame.
pos	a numeric value giving the index of the variable of interest. Additional variables in x are used for grouping according to missingness.
selection	the selection method for grouping according to missingness in multiple additional variables. Possible values are "none" (grouping according to missingness in every other variable that contains missing values), "any" (grouping according to missingness in *any* of the additional variables) and "all" (grouping according to missingness in *all* of the additional variables).
col	a vector of length three giving the colors to be used in the plot. The first two colors are used for the boxplots of the available and missing data, respectively, and the third color for the frequencies of missing values in both variables (see 'Details'). If only one color is supplied, it is used for the boxplots for missing data, whereas the boxplots for the available data are transparent. Else if two colors are supplied, the second one is recycled.
numbers	a logical indicating whether the frequencies of missing values should be displayed (see 'Details').
cex.numbers	the character expansion factor to be used for the frequencies of the missing values.
xlim, ylim	axis limits.
main, sub	main and sub title.
xlab, ylab	axis labels.
axes	a logical indicating whether axes should be drawn on the plot.
frame.plot	a logical indicating whether a box should be drawn around the plot.
labels	either a logical indicating whether labels should be plotted below each box, or a character vector giving the labels.
interactive	a logical indicating whether variables can be switched interactively (see 'Details').
...	for pbox, further arguments and graphical parameters to be passed to boxplot and other functions. For TKRpbox, further arguments to be passed to pbox.
hscale	horizontal scale factor for plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of boxes to be drawn.
vscale	vertical scale factor for the plot to be embedded in a *Tcl/Tk* window (see 'Details').
TKRpar	a list of graphical parameters to be set for the plot to be embedded in a *Tcl/Tk* window (see 'Details' and par).

Details

This plot consists of several boxplots. First, a standard boxplot of the variable of interest is produced. Second, boxplots grouped by observed and missing values according to `selection` are produced for the variable of interest.

Additionally, the frequencies of the missing values can be represented by numbers. If so, the first line corresponds to the observed values of the variable of interest and their distribution in the different groups, the second line to the missing values.

If `interactive=TRUE`, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session.

`TKRpbox` behaves like `pbox` with `selection="none"`, but uses `tkrplot` to embed the plot in a *Tcl/Tk* window. This is useful for drawing a large number of parallel boxes, because scrollbars allow to move from one part of the plot to another.

Value

a list as returned by `boxplot`.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `names` and `cex.text` can still be supplied to `...` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `labels` and `cex.numbers` instead.

Author(s)

Andreas Alfons, Matthias Templ

See Also

`parcoordMiss`

Examples

```
data(chorizonDL, package = "VIM")
pbox(log(chorizonDL[, c(4,5,8,10,11,16:17,19,25,29,37,38,40)]))
```

`prepare` *Transformation and standardization*

Description

This function is used by the `VIM` GUI for transformation and standardization of the data.

Usage

```
prepare(x, scaling = c("none","classical","MCD","robust","onestep"),
        transformation = c("none","minus","reciprocal","logarithm",
            "exponential","boxcox","clr","ilr","alr"),
        alpha = NULL, powers = NULL, start = 0, alrVar)
```

Arguments

`x`	a vector, matrix or `data.frame`.
`scaling`	the scaling to be applied to the data. Possible values are `"none"`, `"classical"`, `MCD`, `"robust"` and `"onestep"`.
`transformation`	the transformation of the data. Possible values are `"none"`, `"minus"`, `"reciprocal"`, `"logarithm"`, `"exponential"`, `"boxcox"`, `"clr"`, `"ilr"` and `"alr"`.
`alpha`	a numeric parameter controlling the size of the subset for the *MCD* (if `scaling="MCD"`). See `covMcd`.
`powers`	a numeric vector giving the powers to be used in the Box-Cox transformation (if `transformation="boxcox"`). If `NULL`, the powers are calculated with function `box.cox.powers`.
`start`	a constant to be added prior to Box-Cox transformation (if `transformation="boxcox"`).
`alrVar`	variable to be used as denominator in the additive logratio transformation (if `transformation="alr"`).

Details

Transformation:

`"none"`: no transformation is used.

`"logarithm"`: compute the the logarithm (to the base 10).

`"boxcox"`: apply a Box-Cox transformation. Powers may be specified or calculated with the function `box.cox.powers`.

Standardization:

`"none"`: no standardization is used.

`"classical"`: apply a z-Transformation on each variable by using function `scale`.

`"robust"`: apply a robustified z-Transformation by using median and MAD.

Value

Transformed and standardized data.

Author(s)

Matthias Templ, modifications by Andreas Alfons

See Also

`scale`, `box.cox.powers`

Examples

```
data(sleep, package = "VIM")
x <- sleep[, c("BodyWgt", "BrainWgt")]
prepare(x, scaling = "robust", transformation = "logarithm")
```

`print.aggr` *Print method for objects of class aggr*

Description

Print method for objects of class `"aggr"`.

Usage

```
## S3 method for class 'aggr'
print(x, digits = NULL, ...)
```

Arguments

`x`	an object of class `"aggr"`.
`digits`	the minimum number of significant digits to be used (see `print.default`).
`...`	further arguments (currently ignored).

Author(s)

Matthias Templ, modifications by Andreas Alfons

See Also

`aggr`

Examples

```
data(sleep, package = "VIM")
a <- aggr(sleep, plot=FALSE)
a
```

`print.summary.aggr` *Print method for objects of class summary.aggr*

Description

Print method for objects of class `"summary.aggr"`.

Usage

```
## S3 method for class 'summary.aggr'
print(x, ...)
```

Arguments

`x`	an object of class `"summary.aggr"`.
`...`	further arguments (currently ignored).

Author(s)

Andreas Alfons

See Also

`summary.aggr`, `aggr`

Examples

```
data(sleep, package = "VIM")
s <- summary(aggr(sleep, plot=FALSE))
s
```

`rugNA` *Rug representation of missing values*

Description

Add a rug representation of missing values in only one of the variables to scatterplots.

Usage

```
rugNA(x, y, ticksize = NULL, side = 1,
      col = "red", alpha = NULL, lwd = 0.5, ...)
```

Arguments

`x, y`	numeric vectors.
`ticksize`	the length of the ticks. Positive lengths give inward ticks.
`side`	an integer giving the side of the plot to draw the rug representation.
`col`	the color to be used for the ticks.
`alpha`	the alpha value (between 0 and 1).
`lwd`	the line width to be used for the ticks.
`...`	further arguments to be passed to `Axis`.

Details

If `side` is 1 or 3, the rug representation consists of values available in `x` but missing in `y`. Else if `side` is 2 or 4, it consists of values available in `y` but missing in `x`.

Author(s)

Andreas Alfons

Examples

```
data(tao, package = "VIM")
x <- tao[, "Air.Temp"]
y <- tao[, "Humidity"]
plot(x, y)
rugNA(x, y, side = 1)
rugNA(x, y, side = 2)
```

`SBS5242` *Synthetic subset of the Austrian structural business statistics data*

Description

Synthetic subset of the Austrian structural business statistics (SBS) data, namely NACE code 52.42 (retail sale of clothing).

Usage

```
data(SBS5242)
```

Details

The Austrian SBS data set consists of more than 320.000 enterprises. Available raw (unedited) data set: 21669 observations in 90 variables, structured according NACE revision 1.1 with 3891 missing values.

We investigate 9 variables of NACE 52.42 (retail sale of clothing).

From these confidential raw data set a non-confidential, close-to-reality, synthetic data set was generated.

Source

http://www.statistik.at

Examples

```
data(SBS5242)
aggr(SBS5242)
```

---

scattJitt *Bivariate jitter plot*

Description

Create a bivariate jitter plot.

Usage

```
scattJitt(x, col = c("skyblue","red","red4"), alpha = NULL,
          cex = par("cex"), col.line = "lightgrey",
          lty = "dashed", lwd = par("lwd"),
          numbers = TRUE, cex.numbers = par("cex"),
          main = NULL, sub = NULL, xlab = NULL,
          ylab = NULL, axes = TRUE, frame.plot = axes,
          labels = c("observed","missing"), ...)
```

Arguments

x	a data.frame or matrix with two columns.
col	a vector of length three giving the colors to be used in the plot. The first color will be used for complete observations, the second color for missing values in only one variable, and the third color for missing values in both variables. If only one color is supplied, it is used for all. Else if two colors are supplied, the second one is recycled.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
cex	the character expansion factor for the plot characters.
col.line	the color for the lines dividing the plot region.
lty	the line type for the lines dividing the plot region (see par).
lwd	the line width for the lines dividing the plot region.
numbers	a logical indicating whether the frequencies of observed/missing values should be displayed (see 'Details').
cex.numbers	the character expansion factor to be used for the frequencies of the observed/missing values.
main, sub	main and sub title.

`xlab, ylab`	axis labels.
`axes`	a logical indicating whether both axes should be drawn on the plot. Use graphical parameter `"xaxt"` or `"yaxt"` to suppress just one of the axes.
`frame.plot`	a logical indicating whether a box should be drawn around the plot.
`labels`	a vector of length two giving the axis labels for the regions for observed/missing values (see 'Details').
`...`	further graphical parameters to be passed down (see `par`).

Details

The amount of observed/missing values is visualized by jittered points. Thereby the plot region is divided into up to four regions according to the existence of missing values in one or both variables. In addition, the amount of observed/missing values can be represented by a number.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the argument `cex.text` can still be supplied to `...` and is handled correctly. Nevertheless, it is deprecated and no longer documented. Use `cex.numbers` instead.

Author(s)

Matthias Templ, modifications by Andreas Alfons

Examples

```
data(tao, package = "VIM")
scattJitt(tao[, c("Air.Temp", "Humidity")])
```

`scattmatrixMiss` *Scatterplot matrix with information about missing values*

Description

Scatterplot matrix in which observations with missing values in certain variables are highlighted.

Usage

```
scattmatrixMiss(x, highlight = NULL,
                selection = c("any","all"), plotvars = NULL,
                col = c("skyblue","red"), alpha = NULL,
                pch = c(1,3), lty = par("lty"),
                diagonal = c("density","none"),
                interactive = TRUE, ...)

TKRscattmatrixMiss(x, highlight = NULL,
```

```
selection = c("any","all"), plotvars = NULL,
col = c("skyblue", "red"), alpha = NULL,
..., hscale = NULL, vscale = NULL,
TKRpar = list())
```

Arguments

x	a matrix or data.frame.
highlight	a vector giving the variables to be used for highlighting. If NULL (the default), all variables are used for highlighting.
selection	the selection method for highlighting missing values in multiple highlight variables. Possible values are "any" (highlighting of missing values in *any* of the highlight variables) and "all" (highlighting of missing values in *all* of the highlight variables).
plotvars	a vector giving the variables to be plotted. If NULL (the default), all variables are plotted.
col	a vector of length two giving the colors to be used in the plot. The second color will be used for highlighting.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
pch	a vector of length two giving the plot characters. The second plot character will be used for the highlighted observations.
lty	a vector of length two giving the line types for the density plots in the diagonal panels (if diagonal="density"). The second line type is used for the highlighted observations. If a single value is supplied, it is used for both non-highlighted and highlighted observations.
diagonal	a character string specifying the plot to be drawn in the diagonal panels. Possible values are "density" (density plots for non-highlighted and highlighted observations) and "none".
interactive	a logical indicating whether the variables to be used for highlighting can be selected interactively (see 'Details').
...	for scattmatrixMiss, further arguments and graphical parameters to be passed to pairsVIM. par("oma") will be set appropriately unless supplied (see par). For TKRscattmatrixMiss, further arguments to be passed to scattmatrixMiss.
hscale	horizontal scale factor for plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of variables.
vscale	vertical scale factor for the plot to be embedded in a *Tcl/Tk* window (see 'Details'). The default value depends on the number of variables.
TKRpar	a list of graphical parameters to be set for the plot to be embedded in a *Tcl/Tk* window (see 'Details' and par).

Details

`scattmatrixMiss` uses `pairsVIM` with a panel function that allows highlighting of missing values.

If `interactive=TRUE`, the variables to be used for highlighting can be selected interactively. Observations with missing values in any or in all of the selected variables are highlighted (as determined by `selection`). A variable can be added to the selection by clicking in a diagonal panel. If a variable is already selected, clicking on the corresponding diagonal panel removes it from the selection. Clicking anywhere else quits the interactive session.

The graphical parameter `oma` will be set unless supplied as an argument.

`TKRscattmatrixMiss` behaves like `scattmatrixMiss`, but uses `tkrplot` to embed the plot in a *Tcl/Tk* window. This is useful if the number of variables is large, because scrollbars allow to move from one part of the plot to another.

Note

Some of the argument names and positions have changed with version 1.3 due to a re-implementation and for more consistency with other plot functions in `VIM`. For back compatibility, the argument `colcomb` can still be supplied to `...` and is handled correctly. Nevertheless, it is deprecated and no longer documented. Use `highlight` instead. The arguments `smooth`, `reg.line` and `legend.plot` are no longer used and ignored if supplied.

Author(s)

Andreas Alfons, Matthias Templ

See Also

`pairsVIM`, `marginmatrix`

Examples

```
data(sleep, package = "VIM")
x <- sleep[, 1:5]
x[,c(1,2,4)] <- log10(x[,c(1,2,4)])
scattmatrixMiss(x, highlight = "Dream")
```

---

scattMiss *Scatterplot with information about missing values*

Description

In addition to a standard scatterplot, lines are plotted for the missing values in one variable.

Usage

```
scattMiss(x, side = 1, col = c("skyblue","red","lightgrey"),
          alpha = NULL, lty = c("dashed","dotted"),
          lwd = par("lwd"), quantiles = c(0.5, 0.975),
          inEllipse = FALSE, zeros = FALSE, xlim = NULL,
          ylim = NULL, main = NULL, sub = NULL, xlab = NULL,
          ylab = NULL, interactive = TRUE, ...)
```

Arguments

x	a matrix or data.frame with two columns.
side	if side=1, a rug representation and vertical lines are plotted for the missing values in the second variable; if side=2, a rug representation and horizontal lines for the missing values in the first variable.
col	a vector of length three giving the colors to be used in the plot. The first color is used for the scatterplot, the second color for the rug representation and the lines, and the third color for the ellipses (see 'Details'). If only one color is supplied, it is used for the scatterplot, the rug representation and the lines, whereas the default color is used for the ellipses. Else if a vector of length two is supplied, the default color is used for the ellipses as well.
alpha	a numeric value between 0 and 1 giving the level of transparency of the colors, or NULL. This can be used to prevent overplotting.
lty	a vector of length two giving the line types for the lines and ellipses. If a single value is supplied, it will be used for both.
lwd	a vector of length two giving the line widths for the lines and ellipses. If a single value is supplied, it will be used for both.
quantiles	a vector giving the quantiles of the chi-square distribution to be used for the tolerance ellipses, or NULL to suppress plotting ellipses (see 'Details').
inEllipse	plot lines only inside the largest ellipse. Ignored if quantiles is NULL.
zeros	a logical vector of length two indicating whether the variables are semi-continuous, i.e., contain a considerable amount of zeros. If TRUE, only the non-zero observations are used for computing the tolerance ellipses. If a single logical is supplied, it is recycled. Ignored if quantiles is NULL.
xlim, ylim	axis limits.
main, sub	main and sub title.
xlab, ylab	axis labels.
interactive	a logical indicating whether the side argument can be changed interactively (see 'Details').
...	further graphical parameters to be passed down (see par).

Details

Information about missing values in one variable is included as vertical or horizontal lines, as determined by the side argument. The lines are thereby drawn at the observed x- or y-value. In

addition, percentage coverage ellipses can be drawn to give a clue about the shape of the bivariate data distribution.

If `interactive`is `TRUE`, clicking in the bottom margin redraws the plot with information about missing values in the first variable and clicking in the left margin redraws the plot with information about missing values in the second variable. Clicking anywhere else in the plot quits the interactive session.

Note

The argument `zeros` has been introduced in version 1.4. As a result, some of the argument positions have changed.

Author(s)

Andreas Alfons

See Also

`marginplot`

Examples

```
data(tao, package = "VIM")
scattMiss(tao[,c("Air.Temp", "Humidity")])
```

`sleep` *Mammal sleep data*

Description

Sleep data with missing values.

Usage

```
data(sleep)
```

Format

A data frame with 62 observations on the following 10 variables.

`BodyWgt` a numeric vector

`BrainWgt` a numeric vector

`NonD` a numeric vector

`Dream` a numeric vector

`Sleep` a numeric vector

`Span` a numeric vector

`Gest` a numeric vector

`Pred` a numeric vector

`Exp` a numeric vector

`Danger` a numeric vector

Source

Allison, T. and Chichetti, D. (1976) Sleep in mammals: ecological and constitutional correlates. *Science* **194 (4266)**, 732–734.

The data set was imported from `GGobi`.

Examples

```
data(sleep, package = "VIM")
summary(sleep)
aggr(sleep)
```

`spineMiss` *Spineplot with information about missing values*

Description

Spineplot or spinogram with highlighting of missing values in other variables by splitting each cell into two parts. Additionally, information about missing values in the variable of interest is shown on the right hand side.

Usage

```
spineMiss(x, pos = 1, selection = c("any", "all"),
          breaks = "Sturges", right = TRUE,
          col = c("skyblue","red","skyblue4","red4"),
          border = NULL, main = NULL, sub = NULL,
          xlab = NULL, ylab = NULL, axes = TRUE,
          labels = axes, only.miss = TRUE,
          miss.labels = axes, interactive = TRUE, ...)
```

Arguments

`x`	a vector, matrix or `data.frame`.
`pos`	a numeric value giving the index of the variable of interest. Additional variables in `x` are used for highlighting.
`selection`	the selection method for highlighting missing values in multiple additional variables. Possible values are `"any"` (highlighting of missing values in *any* of the additional variables) and `"all"` (highlighting of missing values in *all* of the additional variables).

`breaks`	if the variable of interest is numeric, `breaks` controls the breakpoints (see `hist` for possible values).
`right`	logical; if `TRUE` and the variable of interest is numeric, the spinogram cells are right-closed (left-open) intervals.
`col`	a vector of length four giving the colors to be used. If only one color is supplied, the bars are transparent and the supplied color is used for highlighting. Else if two colors are supplied, they are recycled.
`border`	the color to be used for the border of the cells. Use `border=NA` to omit borders.
`main, sub`	main and sub title.
`xlab, ylab`	axis labels.
`axes`	a logical indicating whether axes should be drawn on the plot.
`labels`	if the variable of interest is categorical, either a logical indicating whether labels should be plotted below each cell, or a character vector giving the labels. This is ignored if the variable of interest is numeric.
`only.miss`	logical; if `TRUE`, the missing values in the variable of interest are also visualized by a cell in the spineplot or spinogram. Otherwise, a small spineplot is drawn on the right hand side (see 'Details').
`miss.labels`	either a logical indicating whether label(s) should be plotted below the cell(s) on the right hand side, or a character string or vector giving the label(s) (see 'Details').
`interactive`	a logical indicating whether the variables can be switched interactively (see 'Details').
`...`	further graphical parameters to be passed to `title` and `axis`.

Details

A spineplot is created if the variable of interest is categorial and a spinogram if it is numerical. The horizontal axis is scaled according to relative frequencies of the categories/classes. If more than one variable is supplied, the cells are split according to missingness in the additional variables. Thus the proportion of highlighted observations in each category/class is displayed on the vertical axis. Since the height of each cell corresponds to the proportion of highlighted observations, it is now possible to compare the proportions of missing values among the different categories/classes.

If `only.miss=TRUE`, the missing values in the variable of interest are also visualized by a cell in the spine plot or spinogram. If additional variables are supplied, this cell is again split into two parts according to missingness in the additional variables.

Otherwise, a small spineplot that visualizes missing values in the variable of interest is drawn on the right hand side. The first cell corresponds to observed values and the second cell to missing values. Each of the two cells is again split into two parts according to missingness in the additional variables. Note that this display does not make sense if only one variable is supplied, therefore `only.miss` is ignored in that case.

If `interactive=TRUE`, clicking in the left margin of the plot results in switching to the previous variable and clicking in the right margin results in switching to the next variable. Clicking anywhere else on the graphics device quits the interactive session.

Value

a table containing the frequencies corresponding to the cells.

Note

Some of the argument names and positions have changed with version 1.3 due to extended functionality and for more consistency with other plot functions in `VIM`. For back compatibility, the arguments `xaxlabels` and `missaxlabels` can still be supplied to `...` and are handled correctly. Nevertheless, they are deprecated and no longer documented. Use `labels` and `miss.labels` instead.

The code is based on the function `spineplot` by Achim Zeileis.

Author(s)

Andreas Alfons, Matthias Templ

See Also

`histMiss`, `barMiss`, `mosaicMiss`

Examples

```
data(tao, package = "VIM")
spineMiss(tao[, c("Air.Temp", "Humidity")])
data(sleep, package = "VIM")
spineMiss(sleep[, c("Exp", "Sleep")])
```

`summary.aggr` *Summary method for objects of class aggr*

Description

Summary method for objects of class `"aggr"`.

Usage

```
## S3 method for class 'aggr'
summary(object, ...)
```

Arguments

`object`	an object of class `"aggr"`.
`...`	further arguments.

Value

a list of class `"summary.aggr"` containing the following components:

`missings` a `data.frame` containing the amount of missing values in each variable.

`combinations` a `data.frame` containing a character vector representing the combinations of variables along with their frequencies and percentages.

Author(s)

Matthias Templ, modifications by Andreas Alfons

See Also

`print.summary.aggr`, `aggr`

Examples

```
data(sleep, package = "VIM")
summary(aggr(sleep, plot=FALSE))
```

`tao` *Tropical Atmosphere Ocean (TAO) project data*

Description

A small subsample of the Tropical Atmosphere Ocean (TAO) project data, derived from the `GGOBI` project.

Usage

```
data(tao)
```

Format

A data frame with 736 observations on the following 8 variables.

`Year` a numeric vector

`Latitude` a numeric vector

`Longitude` a numeric vector

`Sea.Surface.Temp` a numeric vector

`Air.Temp` a numeric vector

`Humidity` a numeric vector

`UWind` a numeric vector

`VWind` a numeric vector

Details

All cases recorded for five locations and two time periods.

Source

http://www.pmel.noaa.gov/tao/

Examples

```
data(tao, package = "VIM")
summary(tao)
aggr(tao)
```

---

vmGUImenu *GUI for Visualization and Imputation of Missing Values*

Description

Graphical user interface for visualization and imputation of missing values.

Usage

```
vmGUImenu()
```

Details

The *Data* menu allows to select a data set from the R workspace or load data into the workspace from `RData` files. Furthermore, it can be used to transform variables, which are then appended to the data set in use. Commonly used transformations in official statistics are available, e.g., the Box-Cox transformation and the log-transformation as an important special case of the Box-Cox transformation. In addition, several other transformations that are frequently used for compositional data are implemented. Background maps and coordinates for spatial data can be selected in the data menu as well.

After a data set was chosen, variables can be selected in the main menu, along with a method for scaling. An important feature is that the variables will be used in the same order as they were selected, which is especially useful for parallel coordinate plots. Variables for highlighting are distinguished from the plot variables and can be selected separately. For more than one variable chosen for highlighting, it is possible to select whether observations with missing values in any or in all of these variables should be highlighted.

A plot method can be selected from the *Visualization* menu. Note that plots that are not applicable to the selected variables are disabled, for example, if only one plot variable is selected, multivariate plots cannot be chosen.

Last, but not least, the *Options* menu allows to set the colors and alpha channel to be used in the plots. In addition, it contains an option to embed multivariate plots in `Tcl/Tk` windows. This is useful if the number of observations and/or variables is large, because scrollbars allow to move from one part of the plot to another.

Sections *Imputation* and *Diagonstics* are not yet implemented.

vmGUImenu 57

Author(s)

Andreas Alfons, based on an initial design by Matthias Templ

Index

∗Topic **color**
alphablend, 6
colSequence, 15
rugNA, 44
∗Topic **datasets**
chorizonDL, 9
kola.background, 27
SBS5242, 45
sleep, 51
tao, 55
∗Topic **hplot**
aggr, 3
barMiss, 6
bgmap, 8
colormapMiss, 13
growdotMiss, 17
histMiss, 19
mapMiss, 27
marginmatrix, 29
marginplot, 30
matrixplot, 32
mosaicMiss, 34
pairsVIM, 35
parcoordMiss, 37
pbox, 39
scattJitt, 46
scattmatrixMiss, 47
scattMiss, 49
spineMiss, 52
vmGUImenu, 56
∗Topic **manip**
hotdeck, 22
initialise, 23
irmi, 23
kNN, 25
prepare, 41
∗Topic **multivariate**
vmGUImenu, 56
∗Topic **package**
VIM-package, 2
∗Topic **print**
print.aggr, 43
print.summary.aggr, 44
summary.aggr, 54
∗Topic **utilities**
count, 17

aggr, 3, *43, 44*, *55*
alphablend, 6
Axis, *45*
axis, *7*, *20*, *33*, *53*

barMiss, 6, *21*, *54*
bgmap, 8, *18, 19*, *28*
box.cox.powers, *42*
boxplot, *40, 41*
bubbleFIN, *19*
bubbleMiss, *28*
bubbleMiss *(growdotMiss)*, 17

chorizon, *9*, *12*
chorizonDL, 9
colormapMiss, 13, *19*, *28*
colormapMissLegend *(colormapMiss)*, 13
colSequence, 15, *15*
colSequenceHCL *(colSequence)*, 15
colSequenceRGB *(colSequence)*, 15
count, 17
countInf *(count)*, 17
countNA *(count)*, 17
covMcd, *42*

format, *18*

gowerD *(kNN)*, 25
growdotMiss, *9*, *15*, 17

hex, *13–16*, *33*
hist, *20*, *53*

histMiss, *8*, 19, *54*
hotdeck, 22

iimagMiss *(matrixplot)*, 32
initialise, 23
irmi, 23

kNN, 25
kola.background, 27

labeling_border, *34*
lines, *8*

mapMiss, *9*, *15*, *19*, 27
marginmatrix, 29, *36*, *49*
marginplot, *29*, 30, *30*, *51*
matrixplot, 32
maxCat *(kNN)*, 25
mi, *25*
mosaic, *35*
mosaicMiss, 34, *54*

pairs, *36*
pairsVIM, *29, 30*, 35, *48, 49*
par, *4*, *29*, *31*, *33*, *36*, *38*, *40*, *46–48*, *50*
parcoordMiss, 37, *41*
pbox, 39, *39*
plot.aggr, *4*
plot.aggr *(aggr)*, 3
plot.window, *33*
points, *28*
polarLUV, *13*, *16*, *32*
prepare, 41
print.aggr, *5*, 43
print.default, *43*
print.summary.aggr, 44, *55*

RGB, *13*, *16*, *32*
rugNA, 44

sampleCat *(kNN)*, 25
SBS5242, 45
scale, *42*
scattJitt, 46
scattmatrixMiss, *30*, *36*, 47
scattMiss, *31*, 49
sequential_hcl, *16*
sleep, 51
spineMiss, *8*, *21*, *35*, 52
spineplot, *54*
summary.aggr, *5*, *44*, 54

tao, 55
title, *7*, *20*, *33*, *53*
TKRaggr *(aggr)*, 3
TKRmarginmatrix *(marginmatrix)*, 29
TKRmatrixplot *(matrixplot)*, 32
TKRparcoordMiss *(parcoordMiss)*, 37
TKRpbox *(pbox)*, 39
tkrplot, *5*, *29*, *33*, *39*, *41*, *49*
TKRscattmatrixMiss *(scattmatrixMiss)*, 47

VIM *(VIM-package)*, 2
VIM-package, 2
vmGUImenu, 56

which.minN *(kNN)*, 25

Package 'laeken'

June 6, 2011

Type Package

Title Estimation of indicators on social exclusion and poverty

Version 0.3

Date 2011-06-06

Author Andreas Alfons, Josef Holzer and Matthias Templ

Maintainer Andreas Alfons <andreas.alfons@econ.kuleuven.be>

Depends boot, MASS

Description Estimation of indicators on social exclusion and poverty,as well as Pareto tail modeling for empirical income
distributions (including graphical tools).

License GPL (>= 2)

Repository CRAN

Date/Publication 2011-06-06 18:06:24

R topics documented:

laeken-package 2
arpr 4
arpt 6
bootVar 7
calibVars 9
calibWeights 10
eqInc 12
eqSS 13
eusilc 15
fitPareto 16
gini 18
gpg 20
incMean 23

incMedian . . . 24
incQuintile . . . 25
meanExcessPlot . . . 27
minAMSE . . . 28
paretoQPlot . . . 30
paretoScale . . . 31
paretoTail . . . 32
qsr . . . 35
replaceTail . . . 37
reweightOut . . . 38
rmpg . . . 40
shrinkOut . . . 42
thetaHill . . . 43
thetaISE . . . 44
thetaLS . . . 46
thetaMoment . . . 47
thetaPDC . . . 48
thetaQQ . . . 50
thetaTM . . . 51
thetaWML . . . 53
utils . . . 54
variance . . . 56
weightedMean . . . 58
weightedMedian . . . 59
weightedQuantile . . . 60

Index **62**

`laeken-package` *Estimation of indicators on social exclusion and poverty*

Description

Estimation of indicators on social exclusion and poverty, as well as Pareto tail modeling for empirical income distributions (including graphical tools).

Details

Package:	laeken
Type:	Package
Version:	0.3
Date:	2011-06-06
Depends:	boot, MASS
License:	GPL (>= 2)

Index:

arpr	At-risk-of-poverty rate
arpt	At-risk-of-poverty threshold
bootVar	Bootstrap variance and confidence intervals of indicators on social exclusion and poverty
calibVars	Construct a matrix of binary variables for calibration
calibWeights	Calibrate sample weights
eqInc	Equivalized disposable income
eqSS	Equivalized household size
eusilc	Synthetic EU-SILC survey data
fitPareto	Fit income distribution models with the Pareto distribution
gini	Gini coefficient
gpg	Gender pay (wage) gap.
incMean	Weighted mean income
incMedian	Weighted median income
incQuintile	Weighted income quintile
is.indicator	Utility functions for indicators on social exclusion and poverty
laeken-package	Estimation of indicators on social exclusion and poverty
meanExcessPlot	Mean excess plot
minAMSE	Weighted asymptotic mean squared error (AMSE) estimator
paretoQPlot	Pareto quantile plot
paretoScale	Estimate the scale parameter of a Pareto distribution
paretoTail	Pareto tail modeling for income distributions
qsr	Quintile share ratio
replaceTail	Replace observations under a Pareto model
reweightOut	Reweight outliers in the Pareto model
rmpg	Relative median at-risk-of-poverty gap
shrinkOut	Shrink outliers in the Pareto model
thetaHill	Hill estimator
thetaISE	Integrated squared error (ISE) estimator
thetaLS	Least squares (LS) estimator
thetaMoment	Moment estimator
thetaPDC	Partial density component (PDC) estimator
thetaQQ	QQ-estimator
thetaTM	Trimmed mean estimator
thetaWML	Weighted maximum likelihood estimator
variance	Variance and confidence intervals of indicators on social exclusion and poverty
weightedMean	Weighted mean
weightedMedian	Weighted median
weightedQuantile	Weighted quantiles

Further information is available in the following vignettes:

laeken-pareto	Robust Pareto Tail Modeling for the Estimation of Indicators on Social Exclusion using the R Packag
laeken-standard	Standard Methods for Point Estimation of Indicators on Social Exclusion and Poverty using the R Pa
laeken-variance	Variance Estimation of Indicators on Social Exclusion and Poverty using the R Package laeken (sour

Author(s)

Andreas Alfons, Josef Holzer and Matthias Templ

Maintainer: Andreas Alfons <andreas.alfons@econ.kuleuven.be>

`arpr` *At-risk-of-poverty rate*

Description

Estimate the at-risk-of-poverty rate, which is defined as the proportion of persons with equivalized disposable income below the at-risk-of-poverty threshold.

Usage

```
arpr(inc, weights = NULL, sort = NULL, years = NULL,
     breakdown = NULL, design = NULL, data = NULL, p = 0.6,
     var = NULL, alpha = 0.05, na.rm = FALSE, ...)
```

Arguments

`inc`	either a numeric vector giving the equivalized disposable income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`sort`	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.
`breakdown`	optional; either a numeric vector giving different strata, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, the values for each stratum are computed in addition to the overall value. Note that the same (overall) threshold is used for all strata.

design	optional and only used if var is not NULL; either an integer vector or factor giving different strata for stratified sampling designs, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
data	an optional data.frame.
p	a numeric value in $[0, 1]$ giving the percentage of the weighted median to be used for the at-risk-of-poverty threshold (see arpt).
var	a character string specifying the type of variance estimation to be used, or NULL to omit variance estimation. See variance for possible values.
alpha	numeric; if var is not NULL, this gives the significance level to be used for computing the confidence interval (i.e., the confidence level is $1-$alpha).
na.rm	a logical indicating whether missing values should be removed.
...	if var is not NULL, additional arguments to be passed to variance.

Details

The implementation strictly follows the Eurostat definition.

Value

A list of class "arpr" (which inherits from the class "indicator") with the following components:

value	a numeric vector containing the overall value(s).
valueByStratum	a data.frame containing the values by stratum, or NULL.
varMethod	a character string specifying the type of variance estimation used, or NULL if variance estimation was omitted.
var	a numeric vector containing the variance estimate(s), or NULL.
varByStratum	a data.frame containing the variance estimates by stratum, or NULL.
ci	a numeric vector or matrix containing the lower and upper endpoints of the confidence interval(s), or NULL.
ciByStratum	a data.frame containing the lower and upper endpoints of the confidence intervals by stratum, or NULL.
alpha	a numeric value giving the significance level used for computing the confidence interval(s) (i.e., the confidence level is $1-$alpha), or NULL.
years	a numeric vector containing the different years of the survey.
strata	a character vector containing the different strata of the breakdown.
p	a numeric giving the percentage of the weighted median used for the at-risk-of-poverty threshold.
threshold	a numeric vector containing the at-risk-of-poverty threshold(s).

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat, Luxembourg.

See Also

`arpt`, `variance`

Examples

```
data(eusilc)

# overall value
arpr("eqIncome", weights = "rb050", data = eusilc)

# values by region
arpr("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
```

`arpt` *At-risk-of-poverty threshold*

Description

Estimate the at-risk-of-poverty threshold. The standard definition is to use 60% of the weighted median equivalized disposable income.

Usage

```
arpt(inc, weights = NULL, sort = NULL, years = NULL,
     data = NULL, p = 0.6, na.rm = FALSE)
```

Arguments

`inc`	either a numeric vector giving the equivalized disposable income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`sort`	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.

`data`	an optional `data.frame`.
`p`	a numeric value in $[0, 1]$ giving the percentage of the weighted median to be used for the at-risk-of-poverty threshold.
`na.rm`	a logical indicating whether missing values should be removed.

Details

The implementation strictly follows the Eurostat definition.

Value

A numeric vector containing the value(s) of the at-risk-of-poverty threshold is returned.

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`arpr`, `incMedian`, `weightedMedian`

Examples

```
data(eusilc)
arpt("eqIncome", weights = "rb050", data = eusilc)
```

`bootVar` — *Bootstrap variance and confidence intervals of indicators on social exclusion and poverty*

Description

Compute variance and confidence interval estimates of indicators on social exclusion and poverty based on bootstrap resampling.

Usage

```
bootVar(inc, weights = NULL, years = NULL, breakdown = NULL,
        design = NULL, data = NULL, indicator, R = 100,
        bootType = c("calibrate", "naive"), X, totals = NULL,
        ciType = c("perc", "norm", "basic"),
        alpha = 0.05, seed = NULL, na.rm = FALSE,
        gender = NULL, method = "mean", ...)
```

Arguments

inc	either a numeric vector giving the equivalized disposable income, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
weights	optional; either a numeric vector giving the personal sample weights, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
years	optional; either a numeric vector giving the different years of the survey, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data. If supplied, values are computed for each year.
breakdown	optional; either a numeric vector giving different strata, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data. If supplied, the values for each stratum are computed in addition to the overall value.
design	optional; either an integer vector or factor giving different strata for stratified sampling designs, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data. If supplied, this is used as strata argument in the call to boot.
data	an optional data.frame.
indicator	an object inheriting from the class "indicator" that contains the point estimates of the indicator (see arpr, qsr, rmpg or gini).
R	a numeric value giving the number of bootstrap replicates..
bootType	a character string specifying the type of bootstap to be performed. Possible values are "calibrate" (for calibration of the sample weights of the resampled observations in every iteration) and "naive" (for a naive bootstrap without calibration of the sample weights).
X	if bootType is "calibrate", a matrix of calibration variables.
totals	numeric; if bootType is "calibrate", this gives the population totals. If years is NULL, a vector should be supplied, otherwise a matrix in which each row contains the population totals of the respective year. If this is NULL (the default), the population totals are computed from the sample weights using the Horvitz-Thompson estimator.
ciType	a character string specifying the type of confidence interval(s) to be computed. Possible values are "perc", "norm" and "basic" (see boot.ci).
alpha	a numeric value giving the significance level to be used for computing the confidence interval(s) (i.e., the confidence level is 1−alpha), or NULL.
seed	optional; an integer value to be used as the seed of the random number generator, or an integer vector containing the state of the random number generator to be restored.
na.rm	a logical indicating whether missing values should be removed.
gender	either a numeric vector giving the gender, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.

`method`	mean or median. If weights are provided, the weighted mean or weighted median is estimated.
`...`	if `bootType` is `"calibrate"`, additional arguments to be passed to `calibWeights`.

Value

An object of the same class as `indicator` is returned. See `arpr`, `qsr`, `rmpg` or `gini` for details on the components.

Note

This function gives reasonable variance estimates for basic sample designs such as simple random sampling or stratified simple random sampling.

Author(s)

Andreas Alfons

See Also

`variance`, `calibWeights`, `arpr`, `qsr`, `rmpg`, `gini`

Examples

```
data(eusilc)
a <- arpr("eqIncome", weights = "rb050", data = eusilc)

## naive bootstrap
bootVar("eqIncome", weights = "rb050", design = "db040",
    data = eusilc, indicator = a, bootType = "naive", seed = 123)

## bootstrap with calibration
bootVar("eqIncome", weights = "rb050", design = "db040",
    data = eusilc, indicator = a, X = calibVars(eusilc$db040),
    seed = 123)
```

`calibVars` *Construct a matrix of binary variables for calibration*

Description

Construct a matrix of binary variables for calibration of sample weights according to known marginal population totals.

Usage

```
calibVars(x)
```

Arguments

`x`	a vector that can be interpreted as factor, or a matrix or `data.frame` consisting of such variables.

Value

A matrix of binary variables that indicate membership to the corresponding factor levels.

Author(s)

Andreas Alfons

See Also

`calibWeights`

Examples

```
data(eusilc)
# default method
aux <- calibVars(eusilc$rb090)
head(aux)
# data.frame method
aux <- calibVars(eusilc[, c("db040", "rb090")])
head(aux)
```

`calibWeights` *Calibrate sample weights*

Description

Calibrate sample weights according to known marginal population totals. Based on initial sample weights, the so-called *g*-weights are computed by generalized raking procedures.

Usage

```
calibWeights(X, d, totals, q = NULL,
          method = c("raking", "linear", "logit"),
          bounds = c(0, 10), maxit = 500, tol = 1e-06,
          eps = .Machine$double.eps)
```

Arguments

`X`	a matrix of binary calibration variables (see `calibVars`).
`d`	a numeric vector giving the initial sample weights.
`totals`	a numeric vector of population totals corresponding to the calibration variables in `X`.

q	a numeric vector of positive values accounting for heteroscedasticity. Small values reduce the variation of the *g*-weights.
method	a character string specifying the calibration method to be used. Possible values are "linear" for the linear method, "raking" for the multiplicative method known as raking and "logit" for the logit method.
bounds	a numeric vector of length two giving bounds for the g-weights to be used in the logit method. The first value gives the lower bound (which must be smaller than or equal to 1) and the second value gives the upper bound (which must be larger than or equal to 1).
maxit	a numeric value giving the maximum number of iterations.
tol	the desired accuracy for the iterative procedure.
eps	the desired accuracy for computing the Moore-Penrose generalized inverse (see ginv).

Details

The final sample weights need to be computed by multiplying the resulting *g*-weights with the initial sample weights.

Value

A numeric vector containing the *g*-weights.

Note

This is a faster implementation of parts of calib from package sampling. Note that the default calibration method is raking and that the truncated linear method is not yet implemented.

Author(s)

Andreas Alfons

References

Deville, J.-C. and Särndal, C.-E. (1992) Calibration estimators in survey sampling. *Journal of the American Statistical Association*, **87**(418), 376–382.

Deville, J.-C., Särndal, C.-E. and Sautory, O. (1993) Generalized raking procedures in survey sampling. *Journal of the American Statistical Association*, **88**(423), 1013–1020.

See Also

calibVars, bootVar

Examples

```
data(eusilc)
# construct auxiliary 0/1 variables for genders
aux <- calibVars(eusilc$rb090)
# population totals
totals <- c(3990798, 4191431)
# compute g-weights
g <- calibWeights(aux, eusilc$rb050, totals)
# compute final weights
weights <- g * eusilc$rb050
summary(weights)
```

---

eqInc *Equivalized disposable income*

Description

Compute the equivalized disposable income from household and personal income variables.

Usage

```
eqInc(hid, hplus, hminus, pplus, pminus,
      eqSS, year = NULL, data = NULL)
```

Arguments

hid	if data=NULL, a vector containing the household ID. Otherwise a character string specifying the column of data that contains the household ID.
hplus	if data=NULL, a data.frame containing the household income components that have to be added. Otherwise a character vector specifying the columns of data that contain these income components.
hminus	if data=NULL, a data.frame containing the household income components that have to be subtracted. Otherwise a character vector specifying the columns of data that contain these income components.
pplus	if data=NULL, a data.frame containing the personal income components that have to be added. Otherwise a character vector specifying the columns of data that contain these income components.
pminus	if data=NULL, a data.frame containing the personal income components that have to be subtracted. Otherwise a character vector specifying the columns of data that contain these income components.
eqSS	if data=NULL, a vector containing the equivalized household size. Otherwise a character string specifying the column of data that contains the equivalized household size. See eqSS for more details.
year	if data=NULL, a vector containing the year of the survey. Otherwise a character string specifying the column of data that contains the year.
data	a data.frame containing EU-SILC survey data, or NULL.

Details

All income components should already be imputed, otherwise NAs are simply removed before the calculations.

Value

A numeric vector containing the equivalized disposable income for every individual in `data`.

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`eqSS`

Examples

```
data(eusilc)

# compute a simplified version of the equivalized disposable income
# (not all income components are available in the synthetic data)
hplus <- c("hy040n", "hy050n", "hy070n", "hy080n", "hy090n", "hy110n")
hminus <- c("hy130n", "hy145n")
pplus <- c("py010n", "py050n", "py090n", "py100n",
    "py110n", "py120n", "py130n", "py140n")
eqIncome <- eqInc("db030", hplus, hminus,
    pplus, character(), "eqSS", data=eusilc)

# combine with household ID and equivalized household size
tmp <- cbind(eusilc[, c("db030", "eqSS")], eqIncome)

# show the first 8 rows
head(tmp, 8)
```

`eqSS` *Equivalized household size*

Description

Compute the equivalized household size according to the modified OECD scale adopted in 1994.

Usage

```
eqSS(hid, age, year = NULL, data = NULL)
```

Arguments

`hid`	if `data=NULL`, a vector containing the household ID. Otherwise a character string specifying the column of `data` that contains the household ID.
`age`	if `data=NULL`, a vector containing the age of the individuals. Otherwise a character string specifying the column of `data` that contains the age.
`year`	if `data=NULL`, a vector containing the year of the survey. Otherwise a character string specifying the column of `data` that contains the year.
`data`	a `data.frame` containing EU-SILC survey data, or `NULL`.

Value

A numeric vector containing the equivalized household size for every observation in `data`.

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`eqInc`

Examples

```
data(eusilc)

# calculate equivalized household size
eqSS <- eqSS("db030", "age", data=eusilc)

# combine with household ID and household size
tmp <- cbind(eusilc[, c("db030", "hsize")], eqSS)

# show the first 8 rows
head(tmp, 8)
```

`eusilc` *Synthetic EU-SILC survey data*

Description

This data set is synthetically generated from real Austrian EU-SILC (European Union Statistics on Income and Living Conditions) data.

Usage

```
data(eusilc)
```

Format

A data frame with 14827 observations on the following 28 variables.

`db030` integer; the household ID.

`hsize` integer; the number of persons in the household.

`db040` factor; the federal state in which the household is located (levels `Burgenland`, `Carinthia`, `Lower Austria`, `Salzburg`, `Styria`, `Tyrol`, `Upper Austria`, `Vienna` and `Vorarlberg`).

`rb030` integer; the personal ID.

`age` integer; the person's age.

`rb090` factor; the person's gender (levels `male` and `female`).

`pl030` factor; the person's economic status (levels `1` = working full time, `2` = working part time, `3` = unemployed, `4` = pupil, student, further training or unpaid work experience or in compulsory military or community service, `5` = in retirement or early retirement or has given up business, `6` = permanently disabled or/and unfit to work or other inactive person, `7` = fulfilling domestic tasks and care responsibilities).

`pb220a` factor; the person's citizenship (levels `AT`, `EU` and `Other`).

`py010n` numeric; employee cash or near cash income (net).

`py050n` numeric; cash benefits or losses from self-employment (net).

`py090n` numeric; unemployment benefits (net).

`py100n` numeric; old-age benefits (net).

`py110n` numeric; survivor's benefits (net).

`py120n` numeric; sickness benefits (net).

`py130n` numeric; disability benefits (net).

`py140n` numeric; education-related allowances (net).

`hy040n` numeric; income from rental of a property or land (net).

`hy050n` numeric; family/children related allowances (net).

`hy070n` numeric; housing allowances (net).

`hy080n` numeric; regular inter-household cash transfer received (net).

`hy090n` numeric; interest, dividends, profit from capital investments in unincorporated business (net).

`hy110n` numeric; income received by people aged under 16 (net).

`hy130n` numeric; regular inter-household cash transfer paid (net).

`hy145n` numeric; repayments/receipts for tax adjustment (net).

`eqSS` numeric; the equivalized household size according to the modified OECD scale.

`eqIncome` numeric; a slightly simplified version of the equivalized household income.

`db090` numeric; the household sample weights.

`rb050` numeric; the personal sample weights.

Details

The data set consists of 6000 households and is used in the examples of package `laeken`. Note that this is a synthetic data set based on original EU-SILC survey data.

Only a few of the large number of variables in the original survey are included in this example data set. The variable names are rather cryptic codes, but these are the standardized names used by the statistical agencies. Furthermore, the variables `hsize`, `age`, `eqSS` and `eqIncome` are not included in the standardized format of EU-SILC data, but have been derived from other variables for convenience. Moreover, some very sparse income components were not included in the the generation of this synthetic data set. Thus the equivalized household income is computed from the available income components.

Source

This is a synthetic data set based on Austrian EU-SILC data from 2006. The original sample was provided by Statistics Austria.

References

Eurostat (2004) Description of target variables: Cross-sectional and longitudinal. *EU-SILC 065/04*, Eurostat.

Examples

```
data(eusilc)
summary(eusilc)
```

`fitPareto` *Fit income distribution models with the Pareto distribution*

Description

Fit a Pareto distribution to the upper tail of income data. Since a theoretical distribution is used for the upper tail, this is a semiparametric approach.

Usage

```
fitPareto(x, k = NULL, x0 = NULL, method = "thetaPDC",
          groups = NULL, w = NULL, ...)
```

Arguments

`x`	a numeric vector.
`k`	the number of observations in the upper tail to which the Pareto distribution is fitted.
`x0`	the threshold (scale parameter) above 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 distibution, such as `thetaPDC` (the default). See "Details" for requirements for such a function and "See also" for available functions.
`groups`	an optional 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 every 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.
`w`	an optional numeric vector giving sample weights.
`...`	addtional arguments to be passed to the specified method.

Details

The arguments `k` and `x0` of course correspond with each other. If `k` is supplied, the threshold `x0` is estimated with the $n - k$ largest value in `x`, where n is the number of observations. On the other hand, if the threshold `x0` is supplied, `k` is given by the number of observations in `x` larger than `x0`. Therefore, either `k` or `x0` needs to be supplied. If both are supplied, only `k` is used (mainly for back compatibility).

The function supplied to `method` should take a numeric vector (the observations) as its first argument. If `k` is supplied, it will be passed on (in this case, the function is required to have an argument called `k`). Similarly, if the threshold `x0` is supplied, it will be passed on (in this case, the function is required to have an argument called `x0`). As above, only `k` is passed on if both are supplied. If the function specified by `method` can handle sample weights, the corresponding argument should be called `w`. Additional arguments are passed via the ... argument.

Value

A numeric vector with a Pareto distribution fit to the upper tail.

Note

The arguments `x0` for the threshold (scale parameter) of the Pareto distribution and `w` for sample weights were introduced in version 0.2. This results in slightly different behavior regarding the function calls to `method` compared to prior versions.

18 *gini*

Author(s)

Andreas Alfons and Josef Holzer

See Also

`paretoTail`, `replaceTail`

`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

# using number of observations in tail
eqIncome <- fitPareto(eusilc$eqIncome, k = 175,
    w = eusilc$db090, groups = eusilc$db030)
gini(eqIncome, weights = eusilc$rb050)

# using threshold
eqIncome <- fitPareto(eusilc$eqIncome, x0 = 44150,
    w = eusilc$db090, groups = eusilc$db030)
gini(eqIncome, weights = eusilc$rb050)
```

`gini` *Gini coefficient*

Description

Estimate the Gini coefficient, which is a measure for inequality.

Usage

```
gini(inc, weights = NULL, sort = NULL, years = NULL,
     breakdown = NULL, design = NULL, data = NULL,
     var = NULL, alpha = 0.05, na.rm = FALSE, ...)
```

Arguments

`inc`	either a numeric vector giving the equivalized disposable income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.

`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`sort`	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.
`breakdown`	optional; either a numeric vector giving different strata, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, the values for each stratum are computed in addition to the overall value.
`design`	optional and only used if `var` is not `NULL`; either an integer vector or factor giving different strata for stratified sampling designs, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`data`	an optional `data.frame`.
`var`	a character string specifying the type of variance estimation to be used, or `NULL` to omit variance estimation. See `variance` for possible values.
`alpha`	numeric; if `var` is not `NULL`, this gives the significance level to be used for computing the confidence interval (i.e., the confidence level is `1-alpha`).
`na.rm`	a logical indicating whether missing values should be removed.
`...`	if `var` is not `NULL`, additional arguments to be passed to `variance`.

Details

The implementation strictly follows the Eurostat definition.

Value

A list of class `"gini"` (which inherits from the class `"indicator"`) with the following components:

`value`	a numeric vector containing the overall value(s).
`valueByStratum`	a `data.frame` containing the values by stratum, or `NULL`.
`varMethod`	a character string specifying the type of variance estimation used, or `NULL` if variance estimation was omitted.
`var`	a numeric vector containing the variance estimate(s), or `NULL`.
`varByStratum`	a `data.frame` containing the variance estimates by stratum, or `NULL`.
`ci`	a numeric vector or matrix containing the lower and upper endpoints of the confidence interval(s), or `NULL`.

`ciByStratum`	a `data.frame` containing the lower and upper endpoints of the confidence intervals by stratum, or `NULL`.
`alpha`	a numeric value giving the significance level used for computing the confidence interval(s) (i.e., the confidence level is `1−alpha`), or `NULL`.
`years`	a numeric vector containing the different years of the survey.
`strata`	a character vector containing the different strata of the breakdown.

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`variance`, `qsr`

Examples

```
data(eusilc)

# overall value
gini("eqIncome", weights = "rb050", data = eusilc)

# values by region
gini("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
```

`gpg` *Gender pay (wage) gap.*

Description

Estimate the gender pay (wage) gap.

Usage

```
gpg(inc, gender = NULL, method = c("mean", "median"),
    weights = NULL, sort = NULL, years = NULL, breakdown = NULL,
    design = NULL, data = NULL, var = NULL, alpha = 0.05,
    na.rm = FALSE, ...)
```

Arguments

`inc`	either a numeric vector giving the equivalized disposable income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`gender`	either a factor giving the gender, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`method`	a character string specifying the method to be used. Possible values are `"mean"` for the mean, and `"median"` for the median. If weights are provided, the weighted mean or weighted median is estimated.
`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`sort`	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.
`breakdown`	optional; either a numeric vector giving different strata, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, the values for each stratum are computed in addition to the overall value.
`design`	optional and only used if `var` is not `NULL`; either an integer vector or factor giving different strata for stratified sampling designs, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`data`	an optional `data.frame`.
`var`	a character string specifying the type of variance estimation to be used, or `NULL` to omit variance estimation. See `variance` for possible values.
`alpha`	numeric; if `var` is not `NULL`, this gives the significance level to be used for computing the confidence interval (i.e., the confidence level is `1-alpha`).
`na.rm`	a logical indicating whether missing values should be removed.
`...`	if `var` is not `NULL`, additional arguments to be passed to `variance`.

Details

The implementation strictly follows the Eurostat definition (with default method `"mean"` and alternative method `"median"`). If weights are provided, the weighted mean or weighted median is estimated.

Value

A list of class `"gpg"` (which inherits from the class `"indicator"`) with the following components:

`value`	a numeric vector containing the overall value(s).
`valueByStratum`	a `data.frame` containing the values by stratum, or `NULL`.
`varMethod`	a character string specifying the type of variance estimation used, or `NULL` if variance estimation was omitted.
`var`	a numeric vector containing the variance estimate(s), or `NULL`.
`varByStratum`	a `data.frame` containing the variance estimates by stratum, or `NULL`.
`ci`	a numeric vector or matrix containing the lower and upper endpoints of the confidence interval(s), or `NULL`.
`ciByStratum`	a `data.frame` containing the lower and upper endpoints of the confidence intervals by stratum, or `NULL`.
`alpha`	a numeric value giving the significance level used for computing the confidence interv al(s) (i.e., the confidence level is `1-alpha`), or `NULL`.
`years`	a numeric vector containing the different years of the survey.
`strata`	a character vector containing the different strata of the breakdown.

Author(s)

Matthias Templ and Alexander Haider, using code for breaking down estimation by Andreas Alfons.

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`variance`, `qsr`, `gini`

Examples

```
data(eusilc)
### clearly, children and elder people may not work in Austria:
eusilc <- eusilc[eusilc$age > 17 & eusilc$age < 66, ]
### full time workers:
eusilc <- eusilc[eusilc$pl030 == 1, ]
### employees's cash income:
### py010n
eusilc <- eusilc[!is.na(eusilc$py010n), ]
### for estimation of the GPG, use hourly rates of people
### who earn money and NOT just yearly income as done in
### the following examples!

median_no_breakdown <- gpg("py010n", "rb090", method = "median",
    weights = "rb050", data = eusilc)
mean_no_breakdown <- gpg("py010n", "rb090", method = "mean",
    weights = "rb050", data = eusilc)
```

```
variance("py010n", gender = "rb090", method = "median",
    weights = "rb050", design = "db040", data = eusilc,
    indicator = median_no_breakdown, bootType = "naive",
    seed = 123)

variance("py010n", gender = "rb090", method = "mean",
    weights = "rb050", design = "db040", data = eusilc,
    indicator = mean_no_breakdown, bootType = "naive",
    seed = 123)


median_breakdown_area <- gpg("py010n", "rb090",
    method = "median", breakdown = "db040",
    weights = "rb050", data = eusilc)

variance("py010n", gender = "rb090", method = "median",
    breakdown = "db040", weights = "rb050", design = "db040",
    data = eusilc, indicator = median_breakdown_area,
    bootType = "naive", seed = 123)


mean_breakdown_area <- gpg("py010n", "rb090", method = "mean",
    breakdown = "db040", weights = "rb050", data = eusilc)

variance("py010n", gender = "rb090", method = "mean",
    breakdown = "db040", weights = "rb050", design = "db040",
    data = eusilc, indicator = mean_breakdown_area,
    bootType = "naive", seed = 123)
```

`incMean` *Weighted mean income*

Description

Compute the weighted mean income.

Usage

```
incMean(inc, weights = NULL, years = NULL,
        data = NULL, na.rm = FALSE)
```

Arguments

`inc`	either a numeric vector giving the (equivalized disposable) income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.

years	optional; either a numeric vector giving the different years of the survey, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data. If supplied, values are computed for each year.
data	an optional data.frame.
na.rm	a logical indicating whether missing values should be removed.

Value

A numeric vector containing the value(s) of the weighted mean income is returned.

Author(s)

Andreas Alfons

See Also

weightedMean

Examples

```
data(eusilc)
incMean("eqIncome", weights = "rb050", data = eusilc)
```

---

incMedian *Weighted median income*

Description

Compute the weighted median income.

Usage

```
incMedian(inc, weights = NULL, sort = NULL,
          years = NULL, data = NULL, na.rm = FALSE)
```

Arguments

inc	either a numeric vector giving the (equivalized disposable) income, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
weights	optional; either a numeric vector giving the personal sample weights, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
sort	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.

`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.
`data`	an optional `data.frame`.
`na.rm`	a logical indicating whether missing values should be removed.

Details

The implementation strictly follows the Eurostat definition.

Value

A numeric vector containing the value(s) of the weighted median income is returned.

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`arpt`, `weightedMedian`

Examples

```
data(eusilc)
incMedian("eqIncome", weights = "rb050", data = eusilc)
```

`incQuintile` *Weighted income quintile*

Description

Compute weighted income quintiles.

Usage

```
incQuintile(inc, weights = NULL, sort = NULL, years = NULL,
            k = c(1, 4), data = NULL, na.rm = FALSE)
```

Arguments

inc	either a numeric vector giving the (equivalized disposable) income, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
weights	optional; either a numeric vector giving the personal sample weights, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
sort	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data.
years	optional; either a numeric vector giving the different years of the survey, or (if data is not NULL) a character string, an integer or a logical vector specifying the corresponding column of data. If supplied, values are computed for each year.
k	a vector of integers between 0 and 5 specifying the quintiles to be computed (0 gives the minimum, 5 the maximum).
data	an optional data.frame.
na.rm	a logical indicating whether missing values should be removed.

Details

The implementation strictly follows the Eurostat definition.

Value

A numeric vector (if years is NULL) or matrix (if years is not NULL) containing the values of the weighted income quintiles specified by k are returned.

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

qsr, weightedQuantile

Examples

```
data(eusilc)
incQuintile("eqIncome", weights = "rb050", data = eusilc)
```

`meanExcessPlot` *Mean excess plot*

Description

The Mean Excess plot is a graphical method for detecting the threshold (scale parameter) of a Pareto distribution.

Usage

```
meanExcessPlot(x, w = NULL, probs, interactive = TRUE, ...)
```

Arguments

`x`	a numeric vector.
`w`	an optional numeric vector giving sample weights.
`probs`	an optional numeric vector of probabilities with values in $[0, 1]$, defining the quantiles to be plotted. This is useful for large data sets, when it may not be desirable to plot every single point.
`interactive`	a logical indicating whether the threshold (scale parameter) can be selected interactively by clicking on points. Information on the selected threshold is then printed on the R console.
`...`	additional arguments to be passed to `plot.default`.

Details

The corresponding mean excesses are plotted against the values of `x` (if supplied, only those specified by `probs`). If the tail of the data follows a Pareto distribution, these observations show a positive linear trend. The leftmost point of a fitted line can thus be used as an estimate of the threshold (scale parameter).

The interactive selection of the threshold (scale parameter) is implemented using `identify`. For the usual `X11` device, the selection process is thus terminated by pressing any mouse button other than the first. For the `quartz` device (on Mac OS X systems), the process is terminated either by a secondary click (usually second mouse button or `Ctrl`-click) or by pressing the `ESC` key.

Value

If `interactive` is `TRUE`, the last selection for the threshold is returned invisibly as an object of class `"paretoScale"`, which consists of the following components:

`x0`	the selected threshold (scale parameter).
`k`	the number of observations in the tail (i.e., larger than the threshold).

Note

The functionality to account for sample weights and to select the threshold (scale parameter) interactively was introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

See Also

minAMSE, paretoScale, paretoQPlot, identify

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# with sample weights
meanExcessPlot(eusilc$eqIncome, w = eusilc$db090)

# without sample weights
meanExcessPlot(eusilc$eqIncome)
```

---

minAMSE *Weighted asymptotic mean squared error (AMSE) estimator*

Description

Estimate the scale and shape parameters of a Pareto distribution with an iterative procedure based on minimizing the weighted asymptotic mean squared error (AMSE) of the Hill estimator.

Usage

```
minAMSE(x, weight = c("Bernoulli", "JASA"),
        kmin, kmax, mmax, tol = 0, maxit = 100)

## S3 method for class 'minAMSE'
print(x, ...)
```

Arguments

x	for minAMSE, a numeric vector. The print method is called by the generic function if an object of class "minAMSE" is supplied.
weight	a character vector specifying the weighting scheme to be used in the procedure. If "Bernoulli", the weight functions as described in the *Bernoulli* paper are applied. If "JASA", the weight functions as described in the *Journal of the Americal Statistical Association* are used.
kmin	An optional integer giving the lower bound for finding the optimal number of observations in the tail. It defaults to $[\frac{n}{100}]$, where n denotes the number of observations in x (see the references).

`kmax`	An optional integer giving the upper bound for finding the optimal number of observations in the tail (see "Details").
`mmax`	An optional integer giving the upper bound for finding the optimal number of observations for computing the nuisance parameter ρ (see "Details" and the references).
`tol`	an integer giving the desired tolerance level for finding the optimal number of observations in the tail.
`maxit`	a positive integer giving the maximum number of iterations.
`...`	additional arguments to be passed to `print.default`.

Details

The weights used in the weighted AMSE depend on a nuisance parameter ρ. Both the optimal number of observations in the tail and the nuisance parameter ρ are estimated iteratively using nonlinear integer minimization. This is currently done by a brute force algorithm, hence it is stronly recommended to supply upper bounds `kmax` and `mmax`.

See the references for more details on the iterative algorithm.

Value

An object of class `"minAMSE"` containing the following components:

`kopt`	the optimal number of observations in the tail.
`x0`	the corresponding threshold.
`theta`	the estimated shape parameter of the Pareto distribution.
`MSEmin`	the minimal MSE.
`rho`	the estimated nuisance parameter.
`k`	the examined range for the number of observations in the tail.
`MSE`	the corresponding MSEs.

Author(s)

Josef Holzer and Andreas Alfons

References

Beirlant, J., Vynckier, P. and Teugels, J.L. (1996) Tail index estimation, Pareto quantile plots, and regression diagnostics. *Journal of the American Statistical Association*, **91**(436), 1659–1667.

Beirlant, J., Vynckier, P. and Teugels, J.L. (1996) Excess functions and estimation of the extreme-value index. *Bernoulli*, **2**(4), 293–318.

Dupuis, D.J. and Victoria-Feser, M.-P. (2006) A robust prediction error criterion for Pareto modelling of upper tails. *The Canadian Journal of Statistics*, **34**(4), 639–658.

See Also

`thetaHill`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
minAMSE(eusilc$eqIncome[!duplicated(eusilc$db030)],
    kmin = 50, kmax = 150, mmax = 250)
```

`paretoQPlot` *Pareto quantile plot*

Description

The Pareto quantile plot is a graphical method for inspecting the parameters of a Pareto distribution.

Usage

```
paretoQPlot(x, w = NULL, xlab = NULL, ylab = NULL,
            interactive = TRUE, ...)
```

Arguments

`x`	a numeric vector.
`w`	an optional numeric vector giving sample weights.
`xlab, ylab`	axis labels.
`interactive`	a logical indicating whether the threshold (scale parameter) can be selected interactively by clicking on points. Information on the selected threshold is then printed on the R console.
`...`	additional arguments to be passed to `plot.default`.

Details

If the Pareto model holds, there exists a linear relationship between the lograrithms of the observed values and the quantiles of the standard exponential distribution, since the logarithm of a Pareto distributed random variable follows an exponential distribution. Hence the logarithms of the observed values are plotted against the corresponding theoretical quantiles. If the tail of the data follows a Pareto distribution, these observations form almost a straight line. The leftmost point of a fitted line can thus be used as an estimate of the threshold (scale parameter). The slope of the fitted line is in turn an estimate of $\frac{1}{\theta}$, the reciprocal of the shape parameter.

The interactive selection of the threshold (scale parameter) is implemented using `identify`. For the usual `X11` device, the selection process is thus terminated by pressing any mouse button other than the first. For the `quartz` device (on Mac OS X systems), the process is terminated either by a secondary click (usually second mouse button or `Ctrl`-click) or by pressing the `ESC` key.

Value

If `interactive` is `TRUE`, the last selection for the threshold is returned invisibly as an object of class `"paretoScale"`, which consists of the following components:

`x0`	the selected threshold (scale parameter).
`k`	the number of observations in the tail (i.e., larger than the threshold).

Note

The functionality to account for sample weights and to select the threshold (scale parameter) interactively was introduced in version 0.2. Also starting with version 0.2, a logarithmic y-axis is now used to display the axis labels in the scale of the original values.

Author(s)

Andreas Alfons and Josef Holzer

References

Beirlant, J., Vynckier, P. and Teugels, J.L. (1996) Tail index estimation, Pareto quantile plots, and regression diagnostics. *Journal of the American Statistical Association*, **91**(436), 1659–1667.

See Also

`minAMSE`, `paretoScale`, `meanExcessPlot`, `identify`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# with sample weights
paretoQPlot(eusilc$eqIncome, w = eusilc$db090)

# without sample weights
paretoQPlot(eusilc$eqIncome)
```

---

paretoScale *Estimate the scale parameter of a Pareto distribution*

Description

Estimate the scale parameter of a Pareto distribution, i.e., the threshold for Pareto tail modeling.

Usage

```
paretoScale(x, w = NULL, groups = NULL, method = "vanKerm", na.rm = FALSE)
```

Arguments

x	a numeric vector.
w	an optional numeric vector giving sample weights.
groups	an optional 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 every first group member to appear are used for estimating the threshold (scale parameter).
method	a character string specifying the estimation method. If "vanKerm", van Kerm's method is used, which is a rule of thumb specifically designed for the equivalized disposable income in EU-SILC data (currently the only method implemented).
na.rm	a logical indicating whether missing values in x should be omitted.

Value

An object of class "paretoScale", which consists of the following components:

x0	the threshold (scale parameter).
k	the number of observations in the tail (i.e., larger than the threshold).

Author(s)

Andreas Alfons

References

Van Kerm, P. (2007) Extreme incomes and the estimation of poverty and inequality indicators from EU-SILC. IRISS Working Paper Series 2007-01, CEPS/INSTEAD.

See Also

minAMSE, paretoQPlot, meanExcessPlot

Examples

```
data(eusilc)
paretoScale(eusilc$eqIncome, eusilc$db090, groups = eusilc$db030)
```

---

paretoTail *Pareto tail modeling for income distributions*

Description

Fit a Pareto distribution to the upper tail of income data. Since a theoretical distribution is used for the upper tail, this is a semiparametric approach.

Usage

```
paretoTail(x, k = NULL, x0 = NULL, method = "thetaPDC",
           groups = NULL, w = NULL, alpha = 0.01, ...)
```

Arguments

x	a numeric vector.
k	the number of observations in the upper tail to which the Pareto distribution is fitted.
x0	the threshold (scale parameter) above 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 distibution, such as thetaPDC (the default). See "Details" for requirements for such a function and "See also" for available functions.
groups	an optional 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 every first group member to appear are used for fitting the Pareto distribution.
w	an optional numeric vector giving sample weights.
alpha	numeric; values above the theoretical $1-$alpha quantile of the fitted Pareto distribution will be flagged as outliers for further treatment with reweightOut or replaceOut.
...	addtional arguments to be passed to the specified method.

Details

The arguments k and x0 of course correspond with each other. If k is supplied, the threshold x0 is estimated with the $n - k$ largest value in x, where n is the number of observations. On the other hand, if the threshold x0 is supplied, k is given by the number of observations in x larger than x0. Therefore, either k or x0 needs to be supplied. If both are supplied, only k is used.

The function supplied to method should take a numeric vector (the observations) as its first argument. If k is supplied, it will be passed on (in this case, the function is required to have an argument called k). Similarly, if the threshold x0 is supplied, it will be passed on (in this case, the function is required to have an argument called x0). As above, only k is passed on if both are supplied. If the function specified by method can handle sample weights, the corresponding argument should be called w. Additional arguments are passed via the ... argument.

Value

A list of class "paretoTail" with the following components:

x	the supplied numeric vector.
k	the number of observations in the upper tail to which the Pareto distribution has been fitted.
groups	if supplied, the vector or factor specifying groups of elements.
w	if supplied, the numeric vector of sample weights.

method	the function used to estimate the shape parameter, or the name of the function.
x0	the scale parameter.
theta	the estimated shape parameter.
tail	if groups is not NULL, this gives the groups with values larger than the threshold (scale parameter), otherwise the indices of observations in the upper tail.
alpha	the tuning parameter alpha used for flagging outliers.
out	if groups is not NULL, this gives the groups that are flagged as outliers, otherwise the indices of the flagged observations.

Author(s)

Andreas Alfons

See Also

reweightOut, shrinkOut, replaceOut, replaceTail, fitPareto

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

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090,
    groups = eusilc$db030)

# estimate shape parameter
fit <- paretoTail(eusilc$eqIncome, k = ts$k,
    w = eusilc$db090, groups = eusilc$db030)

# calibration of outliers
w <- reweightOut(fit, calibVars(eusilc$db040))
gini(eusilc$eqIncome, w)

# winsorization of outliers
eqIncome <- shrinkOut(fit)
gini(eqIncome, weights = eusilc$rb050)

# replacement of outliers
eqIncome <- replaceOut(fit)
gini(eqIncome, weights = eusilc$rb050)

# replacement of whole tail
```

```
eqIncome <- replaceTail(fit)
gini(eqIncome, weights = eusilc$rb050)
```

`qsr` *Quintile share ratio*

Description

Estimate the quintile share ratio, which is defined as the ratio of the sum of equivalized disposable income received by the top 20% to the sum of equivalized disposable income received by the bottom 20%.

Usage

```
qsr(inc, weights = NULL, sort = NULL, years = NULL,
    breakdown = NULL, design = NULL, data = NULL,
    var = NULL, alpha = 0.05, na.rm = FALSE, ...)
```

Arguments

`inc`	either a numeric vector giving the equivalized disposable income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`sort`	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.
`breakdown`	optional; either a numeric vector giving different strata, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, the values for each stratum are computed in addition to the overall value.
`design`	optional and only used if `var` is not `NULL`; either an integer vector or factor giving different strata for stratified sampling designs, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`data`	an optional `data.frame`.
`var`	a character string specifying the type of variance estimation to be used, or `NULL` to omit variance estimation. See `variance` for possible values.

alpha	numeric; if var is not NULL, this gives the significance level to be used for computing the confidence interval (i.e., the confidence level is 1−alpha).
na.rm	a logical indicating whether missing values should be removed.
...	if var is not NULL, additional arguments to be passed to variance.

Details

The implementation strictly follows the Eurostat definition.

Value

A list of class "qsr" (which inherits from the class "indicator") with the following components:

value	a numeric vector containing the overall value(s).
valueByStratum	a data.frame containing the values by stratum, or NULL.
varMethod	a character string specifying the type of variance estimation used, or NULL if variance estimation was omitted.
var	a numeric vector containing the variance estimate(s), or NULL.
varByStratum	a data.frame containing the variance estimates by stratum, or NULL.
ci	a numeric vector or matrix containing the lower and upper endpoints of the confidence interval(s), or NULL.
ciByStratum	a data.frame containing the lower and upper endpoints of the confidence intervals by stratum, or NULL.
alpha	a numeric value giving the significance level used for computing the confidence interval(s) (i.e., the confidence level is 1−alpha), or NULL.
years	a numeric vector containing the different years of the survey.
strata	a character vector containing the different strata of the breakdown.

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

incQuintile, variance, gini

Examples

```
data(eusilc)

# overall value
qsr("eqIncome", weights = "rb050", data = eusilc)

# values by region
qsr("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
```

`replaceTail` *Replace observations under a Pareto model*

Description

Replace observations under a Pareto model for the upper tail with values drawn from the fitted distribution.

Usage

```
replaceTail(x, ...)

## S3 method for class 'paretoTail'
replaceTail(x, all = TRUE, ...)

replaceOut(x, ...)
```

Arguments

`x`	an object of class `"paretoTail"` (see `paretoTail`).
`all`	a logical indicating whether all observations in the upper tail should be replaced or only those flagged as outliers.
`...`	additional arguments to be passed down.

Details

`replaceOut(x, ...)` is a simple wrapper for `replaceTail(x, all = FALSE, ...)`.

Value

A numeric vector consisting mostly of the original values, but with observations in the upper tail replaced with values from the fitted Pareto distribution.

Author(s)

Andreas Alfons

See Also

paretoTail, reweightOut, shrinkOut

Examples

```
data(eusilc)


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


## gini coefficient with Pareto tail modeling

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090,
    groups = eusilc$db030)

# estimate shape parameter
fit <- paretoTail(eusilc$eqIncome, k = ts$k,
    w = eusilc$db090, groups = eusilc$db030)

# replacement of outliers
eqIncome <- replaceOut(fit)
gini(eqIncome, weights = eusilc$rb050)

# replacement of whole tail
eqIncome <- replaceTail(fit)
gini(eqIncome, weights = eusilc$rb050)
```

`reweightOut` *Reweight outliers in the Pareto model*

Description

Reweight observations that are flagged as outliers in a Pareto model for the upper tail of the distribution.

Usage

```
reweightOut(x, ...)

## S3 method for class 'paretoTail'
reweightOut(x, X, w = NULL, ...)
```

Arguments

`x`	an object of class `"paretoTail"` (see `paretoTail`).
`X`	a matrix of binary calibration variables (see `calibVars`). This is only used if `x` contains sample weights or if `w` is supplied.
`w`	a numeric vector of sample weights. This is only used if `x` does not contain sample weights, i.e., if sample weights were not considered in estimating the shape parameter of the Pareto distribution.
`...`	additional arguments to be passed down.

Details

If the data contain sample weights, the weights of the outlying observations are set to 1 and the weights of the remaining observations are calibrated according to auxiliary variables. Otherwise, weight 0 is assigned to outliers and weight 1 to other observations.

Value

If the data contain sample weights, a numeric containing the recalibrated weights is returned, otherwise a numeric vector assigning weight 0 to outliers and weight 1 to other observations.

Author(s)

Andreas Alfons

See Also

`paretoTail`, `shrinkOut` , `replaceOut`, `replaceTail`

Examples

```
data(eusilc)

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

## gini coefficient with Pareto tail modeling
# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090,
    groups = eusilc$db030)
# estimate shape parameter
fit <- paretoTail(eusilc$eqIncome, k = ts$k,
    w = eusilc$db090, groups = eusilc$db030)
# calibration of outliers
w <- reweightOut(fit, calibVars(eusilc$db040))
gini(eusilc$eqIncome, w)
```

`rmpg` *Relative median at-risk-of-poverty gap*

Description

Estimate the relative median at-risk-of-poverty gap, which is defined as the relative difference between the median equivalized disposable income of persons below the at-risk-of-poverty threshold and the at-risk-of-poverty threshold itself (expressed as a percentage of the at-risk-of-poverty threshold).

Usage

```
rmpg(inc, weights = NULL, sort = NULL, years = NULL,
     breakdown = NULL, design = NULL, data = NULL,
     var = NULL, alpha = 0.05, na.rm = FALSE, ...)
```

Arguments

`inc`	either a numeric vector giving the equivalized disposable income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`sort`	optional; either a numeric vector giving the personal IDs to be used as tie-breakers for sorting, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.
`breakdown`	optional; either a numeric vector giving different strata, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, the values for each stratum are computed in addition to the overall value. Note that the same (overall) threshold is used for all strata.
`design`	optional and only used if `var` is not `NULL`; either an integer vector or factor giving different strata for stratified sampling designs, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`data`	an optional `data.frame`.
`var`	a character string specifying the type of variance estimation to be used, or `NULL` to omit variance estimation. See `variance` for possible values.
`alpha`	numeric; if `var` is not `NULL`, this gives the significance level to be used for computing the confidence interval (i.e., the confidence level is `1-alpha`).
`na.rm`	a logical indicating whether missing values should be removed.
`...`	if `var` is not `NULL`, additional arguments to be passed to `variance`.

Details

The implementation strictly follows the Eurostat definition.

Value

A list of class `"rmpg"` (which inherits from the class `"indicator"`) with the following components:

`value`	a numeric vector containing the overall value(s).
`valueByStratum`	a `data.frame` containing the values by stratum, or `NULL`.
`varMethod`	a character string specifying the type of variance estimation used, or `NULL` if variance estimation was omitted.
`var`	a numeric vector containing the variance estimate(s), or `NULL`.
`varByStratum`	a `data.frame` containing the variance estimates by stratum, or `NULL`.
`ci`	a numeric vector or matrix containing the lower and upper endpoints of the confidence interval(s), or `NULL`.
`ciByStratum`	a `data.frame` containing the lower and upper endpoints of the confidence intervals by stratum, or `NULL`.
`alpha`	a numeric value giving the significance level used for computing the confidence interval(s) (i.e., the confidence level is $1-$`alpha`), or `NULL`.
`years`	a numeric vector containing the different years of the survey.
`strata`	a character vector containing the different strata of the breakdown.
`threshold`	a numeric vector containing the at-risk-of-poverty threshold(s).

Author(s)

Andreas Alfons

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`arpt`, `variance`

Examples

```
data(eusilc)

# overall value
rmpg("eqIncome", weights = "rb050", data = eusilc)

# values by region
rmpg("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
```

`shrinkOut` *Shrink outliers in the Pareto model*

Description

Shrink observations that are flagged as outliers in a Pareto model for the upper tail of the distribution to the theoretical quantile used for outlier detection.

Usage

```
shrinkOut(x, ...)

## S3 method for class 'paretoTail'
shrinkOut(x, ...)
```

Arguments

`x`	an object of class `"paretoTail"` (see `paretoTail`).
`...`	additional arguments to be passed down (currently ignored as there are no additional arguments in the only method implemented).

Value

A numeric vector consisting mostly of the original values, but with outlying observations in the upper tail shrunken to the corresponding theoretical quantile of the fitted Pareto distribution.

Author(s)

Andreas Alfons

See Also

`paretoTail`, `reweightOut`, `replaceOut`, `replaceTail`

Examples

```
data(eusilc)

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

## gini coefficient with Pareto tail modeling
# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090,
    groups = eusilc$db030)
# estimate shape parameter
fit <- paretoTail(eusilc$eqIncome, k = ts$k,
    w = eusilc$db090, groups = eusilc$db030)
# shrink outliers
```

```
eqIncome <- shrinkOut(fit)
gini(eqIncome, weights = eusilc$rb050)
```

`thetaHill` *Hill estimator*

Description

The Hill estimator uses the maximum likelihood principle to estimate the shape parameter of a Pareto distribution.

Usage

```
thetaHill(x, k = NULL, x0 = NULL, w = NULL)
```

Arguments

`x`	a numeric vector.
`k`	the number of observations in the upper tail to which the Pareto distribution is fitted.
`x0`	the threshold (scale parameter) above which the Pareto distribution is fitted.
`w`	an optional numeric vector giving sample weights.

Details

The arguments `k` and `x0` of course correspond with each other. If `k` is supplied, the threshold `x0` is estimated with the $n - k$ largest value in `x`, where n is the number of observations. On the other hand, if the threshold `x0` is supplied, `k` is given by the number of observations in `x` larger than `x0`. Therefore, either `k` or `x0` needs to be supplied. If both are supplied, only `k` is used (mainly for back compatibility).

Value

The estimated shape parameter.

Note

The arguments `x0` for the threshold (scale parameter) of the Pareto distribution and `w` for sample weights were introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Hill, B.M. (1975) A simple general approach to inference about the tail of a distribution. *The Annals of Statistics*, **3**(5), 1163–1174.

See Also

paretoTail, fitPareto, thetaPDC, thetaWML, thetaISE, minAMSE

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaHill(eusilc$eqIncome, k = ts$k, w = eusilc$db090)

# using threshold
thetaHill(eusilc$eqIncome, x0 = ts$x0, w = eusilc$db090)
```

---

thetaISE *Integrated squared error (ISE) estimator*

Description

The integrated squared error (ISE) estimator estimates the shape parameter of a Pareto distribution based on the relative excesses of observations above a certain threshold.

Usage

```
thetaISE(x, k = NULL, x0 = NULL, w = NULL, ...)
```

Arguments

x	a numeric vector.
k	the number of observations in the upper tail to which the Pareto distribution is fitted.
x0	the threshold (scale parameter) above which the Pareto distribution is fitted.
w	an optional numeric vector giving sample weights.
...	additional arguments to be passed to optimize (see "Details").

Details

The arguments k and x0 of course correspond with each other. If k is supplied, the threshold x0 is estimated with the $n - k$ largest value in x, where n is the number of observations. On the other hand, if the threshold x0 is supplied, k is given by the number of observations in x larger than x0. Therefore, either k or x0 needs to be supplied. If both are supplied, only k is used (mainly for back compatibility).

The ISE estimator minimizes the integrated squared error (ISE) criterion with a complete density model. The minimization is carried out using

`optimize`.

Value

The estimated shape parameter.

Note

The arguments `x0` for the threshold (scale parameter) of the Pareto distribution and `w` for sample weights were introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Vandewalle, B., Beirlant, J., Christmann, A., and Hubert, M. (2007) A robust estimator for the tail index of Pareto-type distributions. *Computational Statistics & Data Analysis*, **51**(12), 6252–6268.

See Also

`paretoTail`, `fitPareto`, `thetaPDC`, `thetaHill`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaISE(eusilc$eqIncome, k = ts$k, w = eusilc$db090)

# using threshold
thetaISE(eusilc$eqIncome, x0 = ts$x0, w = eusilc$db090)
```

`thetaLS` *Least squares (LS) estimator*

Description

Estimate the shape parameter of a Pareto distribution using a least squares (LS) approach.

Usage

```
thetaLS(x, k = NULL, x0 = NULL)
```

Arguments

`x`	a numeric vector.
`k`	the number of observations in the upper tail to which the Pareto distribution is fitted.
`x0`	the threshold (scale parameter) above which the Pareto distribution is fitted.

Details

The arguments `k` and `x0` of course correspond with each other. If `k` is supplied, the threshold `x0` is estimated with the $n - k$ largest value in `x`, where n is the number of observations. On the other hand, if the threshold `x0` is supplied, `k` is given by the number of observations in `x` larger than `x0`. Therefore, either `k` or `x0` needs to be supplied. If both are supplied, only `k` is used (mainly for back compatibility).

Value

The estimated shape parameter.

Note

The argument `x0` for the threshold (scale parameter) of the Pareto distribution was introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Brazauskas, V. and Serfling, R. (2000) Robust estimation of tail parameters for two-parameter Pareto and exponential models via generalized quantile statistics. *Extremes*, **3**(3), 231–249.

Brazauskas, V. and Serfling, R. (2000) Robust and efficient estimation of the tail index of a single-parameter Pareto distribution. *North American Actuarial Journal*, **4**(4), 12–27.

See Also

paretoTail, fitPareto

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaLS(eusilc$eqIncome, k = ts$k)

# using threshold
thetaLS(eusilc$eqIncome, x0 = ts$x0)
```

---

thetaMoment *Moment estimator*

Description

Estimate the shape parameter of a Pareto distribution based on moments.

Usage

```
thetaMoment(x, k = NULL, x0 = NULL)
```

Arguments

x	a numeric vector.
k	the number of observations in the upper tail to which the Pareto distribution is fitted.
x0	the threshold (scale parameter) above which the Pareto distribution is fitted.

Details

The arguments k and x0 of course correspond with each other. If k is supplied, the threshold x0 is estimated with the $n - k$ largest value in x, where n is the number of observations. On the other hand, if the threshold x0 is supplied, k is given by the number of observations in x larger than x0. Therefore, either k or x0 needs to be supplied. If both are supplied, only k is used (mainly for back compatibility).

Value

The estimated shape parameter.

Note

The argument `x0` for the threshold (scale parameter) of the Pareto distribution was introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Dekkers, A.L.M., Einmahl, J.H.J. and de Haan, L. (1989) A moment estimator for the index of an extreme-value distribution. *The Annals of Statistics*, **17**(4), 1833–1855.

See Also

`paretoTail`, `fitPareto`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaMoment(eusilc$eqIncome, k = ts$k)

# using threshold
thetaMoment(eusilc$eqIncome, x0 = ts$x0)
```

`thetaPDC` *Partial density component (PDC) estimator*

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 = NULL, x0 = NULL, w = NULL, ...)
```

Arguments

`x`	a numeric vector.
`k`	the number of observations in the upper tail to which the Pareto distribution is fitted.
`x0`	the threshold (scale parameter) above which the Pareto distribution is fitted.
`w`	an optional numeric vector giving sample weights.
`...`	additional arguments to be passed to `optimize` (see "Details").

Details

The arguments `k` and `x0` of course correspond with each other. If `k` is supplied, the threshold `x0` is estimated with the $n - k$ largest value in `x`, where n is the number of observations. On the other hand, if the threshold `x0` is supplied, `k` is given by the number of observations in `x` larger than `x0`. Therefore, either `k` or `x0` needs to be supplied. If both are supplied, only `k` is used (mainly for back compatibility).

The PDC estimator minimizes the integrated squared error (ISE) criterion with an incomplete density mixture model. The minimization is carried out using

`optimize`.

Value

The estimated shape parameter.

Note

The arguments `x0` for the threshold (scale parameter) of the Pareto distribution and `w` for sample weights were introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Vandewalle, B., Beirlant, J., Christmann, A., and Hubert, M. (2007) A robust estimator for the tail index of Pareto-type distributions. *Computational Statistics & Data Analysis*, **51**(12), 6252–6268.

See Also

`paretoTail`, `fitPareto`, `thetaISE`, `thetaHill`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]
```

```
# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaPDC(eusilc$eqIncome, k = ts$k, w = eusilc$db090)

# using threshold
thetaPDC(eusilc$eqIncome, x0 = ts$x0, w = eusilc$db090)
```

`thetaQQ` *QQ-estimator*

Description

Estimate the shape parameter of a Pareto distribution using a quantile-quantile approach.

Usage

```
thetaQQ(x, k = NULL, x0 = NULL)
```

Arguments

`x`	a numeric vector.
`k`	the number of observations in the upper tail to which the Pareto distribution is fitted.
`x0`	the threshold (scale parameter) above which the Pareto distribution is fitted.

Details

The arguments `k` and `x0` of course correspond with each other. If `k` is supplied, the threshold `x0` is estimated with the $n - k$ largest value in `x`, where n is the number of observations. On the other hand, if the threshold `x0` is supplied, `k` is given by the number of observations in `x` larger than `x0`. Therefore, either `k` or `x0` needs to be supplied. If both are supplied, only `k` is used (mainly for back compatibility).

Value

The estimated shape parameter.

Note

The argument `x0` for the threshold (scale parameter) of the Pareto distribution was introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Kratz, M.F. and Resnick, S.I. (1996) The QQ-estimator and heavy tails. *Stochastic Models*, **12**(4), 699–724.

See Also

`paretoTail`, `fitPareto`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaQQ(eusilc$eqIncome, k = ts$k)

# using threshold
thetaQQ(eusilc$eqIncome, x0 = ts$x0)
```

`thetaTM` *Trimmed mean estimator*

Description

Estimate the shape parameter of a Pareto distribution using a trimmed mean approach.

Usage

```
thetaTM(x, k = NULL, x0 = NULL, beta = 0.05)
```

Arguments

`x`	a numeric vector.
`k`	the number of observations in the upper tail to which the Pareto distribution is fitted.
`x0`	the threshold (scale parameter) above which the Pareto distribution is fitted.
`beta`	A numeric vector of length two giving the trimming proportions for the lower and upper end of the tail, respectively. If a single numeric value is supplied, it is recycled.

Details

The arguments `k` and `x0` of course correspond with each other. If `k` is supplied, the threshold `x0` is estimated with the $n - k$ largest value in `x`, where n is the number of observations. On the other hand, if the threshold `x0` is supplied, `k` is given by the number of observations in `x` larger than `x0`. Therefore, either `k` or `x0` needs to be supplied. If both are supplied, only `k` is used (mainly for back compatibility).

Value

The estimated shape parameter.

Note

The argument `x0` for the threshold (scale parameter) of the Pareto distribution was introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Brazauskas, V. and Serfling, R. (2000) Robust estimation of tail parameters for two-parameter Pareto and exponential models via generalized quantile statistics. *Extremes*, **3**(3), 231–249.

Brazauskas, V. and Serfling, R. (2000) Robust and efficient estimation of the tail index of a single-parameter Pareto distribution. *North American Actuarial Journal*, **4**(4), 12–27.

See Also

`paretoTail`, `fitPareto`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaTM(eusilc$eqIncome, k = ts$k)

# using threshold
thetaTM(eusilc$eqIncome, x0 = ts$x0)
```

`thetaWML` *Weighted maximum likelihood estimator*

Description

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

Usage

```
thetaWML(x, k = NULL, x0 = NULL,
         weight = c("residuals", "probability"),
         const, bias = TRUE, ...)
```

Arguments

`x`	a numeric vector.
`k`	the number of observations in the upper tail to which the Pareto distribution is fitted.
`x0`	the threshold (scale parameter) above 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 references for more details.
`bias`	a logical indicating whether bias correction should be applied.
`...`	additional arguments to be passed to `uniroot` (see "Details").

Details

The arguments `k` and `x0` of course correspond with each other. If `k` is supplied, the threshold `x0` is estimated with the $n - k$ largest value in `x`, where n is the number of observations. On the other hand, if the threshold `x0` is supplied, `k` is given by the number of observations in `x` larger than `x0`. Therefore, either `k` or `x0` needs to be supplied. If both are supplied, only `k` is used (mainly for back compatibility).

The weighted maximum likelihood estimator belongs to the class of M-estimators. In order to obtain the estimate, the root of a certain function needs to be found, which is implemented using `uniroot`.

Value

The estimated shape parameter.

Note

The argument `x0` for the threshold (scale parameter) of the Pareto distribution was introduced in version 0.2.

Author(s)

Andreas Alfons and Josef Holzer

References

Dupuis, D.J. and Morgenthaler, S. (2002) Robust weighted likelihood estimators with an application to bivariate extreme value problems. *The Canadian Journal of Statistics*, **30**(1), 17–36.

Dupuis, D.J. and Victoria-Feser, M.-P. (2006) A robust prediction error criterion for Pareto modelling of upper tails. *The Canadian Journal of Statistics*, **34**(4), 639–658.

See Also

`paretoTail`, `fitPareto`

Examples

```
data(eusilc)
# equivalized disposable income is equal for each household
# member, therefore only one household member is taken
eusilc <- eusilc[!duplicated(eusilc$db030),]

# estimate threshold
ts <- paretoScale(eusilc$eqIncome, w = eusilc$db090)

# using number of observations in tail
thetaWML(eusilc$eqIncome, k = ts$k)

# using threshold
thetaWML(eusilc$eqIncome, x0 = ts$x0)
```

`utils` *Utility functions for indicators on social exclusion and poverty*

Description

Test for class, print and take subsets of indicators on social exclusion and poverty.

Usage

```
is.indicator(x)

is.arpr(x)

is.qsr(x)

is.rmpg(x)

is.gini(x)

is.gpg(x)

## S3 method for class 'indicator'
print(x, ...)

## S3 method for class 'arpr'
print(x, ...)

## S3 method for class 'rmpg'
print(x, ...)

## S3 method for class 'indicator'
subset(x, years = NULL, strata = NULL, ...)

## S3 method for class 'arpr'
subset(x, years = NULL, strata = NULL, ...)

## S3 method for class 'rmpg'
subset(x, years = NULL, strata = NULL, ...)
```

Arguments

`x`	for `is.xyz`, any object to be tested. The `print` and `subset` methods are called by the generic functions if an object of the respective class is supplied.
`years`	an optional numeric vector giving the years to be extracted.
`strata`	an optional vector giving the strata of the breakdown to be extracted.
`...`	additional arguments to be passed to and from methods.

Value

`is.indicator` returns `TRUE` if `x` inherits from class `"indicator"` and `FALSE` otherwise.

`is.arpr` returns `TRUE` if `x` inherits from class `"arpr"` and `FALSE` otherwise.

`is.qsr` returns `TRUE` if `x` inherits from class `"qsr"` and `FALSE` otherwise.

`is.rmpg` returns `TRUE` if `x` inherits from class `"rmpg"` and `FALSE` otherwise.

`is.gini` returns `TRUE` if `x` inherits from class `"gini"` and `FALSE` otherwise.

`is.gini` returns `TRUE` if `x` inherits from class `"gini"` and `FALSE` otherwise.

`print.indicator`, `print.arpr` and `print.rmpg` return `x` invisibly.

`subset.indicator`, `subset.arpr` and `subset.rmpg` return a subset of `x` of the same class.

See Also

`arpr`, `qsr`, `rmpg`, `gini`, `gpg`

Examples

```
data(eusilc)

# at-risk-of-poverty rate
a <- arpr("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
print(a)
is.arpr(a)
is.indicator(a)
subset(a, strata = c("Lower Austria", "Vienna"))

# quintile share ratio
q <- qsr("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
print(q)
is.qsr(q)
is.indicator(q)
subset(q, strata = c("Lower Austria", "Vienna"))

# relative median at-risk-of-poverty gap
r <- rmpg("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
print(r)
is.rmpg(r)
is.indicator(r)
subset(r, strata = c("Lower Austria", "Vienna"))

# Gini coefficient
g <- gini("eqIncome", weights = "rb050",
    breakdown = "db040", data = eusilc)
print(g)
is.gini(g)
is.indicator(g)
subset(g, strata = c("Lower Austria", "Vienna"))
```

`variance` *Variance and confidence intervals of indicators on social exclusion and poverty*

Description

Compute variance and confidence interval estimates of indicators on social exclusion and poverty.

Usage

```
variance(inc, weights = NULL, years = NULL, breakdown = NULL,
         design = NULL, data = NULL, indicator, alpha = 0.05,
         na.rm = FALSE, type = "bootstrap", gender = NULL, method = "mean", ...)
```

Arguments

`inc`	either a numeric vector giving the equivalized disposable income, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`weights`	optional; either a numeric vector giving the personal sample weights, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`years`	optional; either a numeric vector giving the different years of the survey, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, values are computed for each year.
`breakdown`	optional; either a numeric vector giving different strata, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`. If supplied, the values for each stratum are computed in addition to the overall value.
`design`	optional; either an integer vector or factor giving different strata for stratified sampling designs, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`data`	an optional `data.frame`.
`indicator`	an object inheriting from the class `"indicator"` that contains the point estimates of the indicator (see `arpr`, `qsr`, `rmpg` or `gini`).
`alpha`	a numeric value giving the significance level to be used for computing the confidence interval(s) (i.e., the confidence level is `1-alpha`), or `NULL`.
`na.rm`	a logical indicating whether missing values should be removed.
`type`	a character string specifying the type of variance estimation to be used. Currently, only `"bootstrap"` is implemented for variance estimation based on bootstrap resampling (see `bootVar`).
`gender`	either a numeric vector giving the gender, or (if `data` is not `NULL`) a character string, an integer or a logical vector specifying the corresponding column of `data`.
`method`	mean or median. If weights are provided, the weighted mean or weighted median is estimated.
`...`	additional arguments to be passed to `bootVar`.

Details

This is a wrapper function for computing variance and confidence interval estimates of indicators on social exclusion and poverty.

Value

An object of the same class as `indicator` is returned. See `arpr`, `qsr`, `rmpg` or `gini` for details on the components.

Author(s)

Andreas Alfons

See Also

`bootVar`, `arpr`, `qsr`, `rmpg`, `gini`

Examples

```
data(eusilc)
a <- arpr("eqIncome", weights = "rb050", data = eusilc)

## naive bootstrap
variance("eqIncome", weights = "rb050", design = "db040",
    data = eusilc, indicator = a, bootType = "naive", seed = 123)

## bootstrap with calibration
variance("eqIncome", weights = "rb050", design = "db040",
    data = eusilc, indicator = a, X = calibVars(eusilc$db040),
    seed = 123)
```

`weightedMean` *Weighted mean*

Description

Compute the weighted mean.

Usage

```
weightedMean(x, weights = NULL, na.rm = FALSE)
```

Arguments

`x`	a numeric vector.
`weights`	an optional numeric vector giving the sample weights.
`na.rm`	a logical indicating whether missing values in `x` should be omitted.

Details

This is a simple wrapper function calling `weighted.mean` if sample weights are supplied and `mean` otherwise.

Value

The weighted mean of values in `x` is returned.

Author(s)

Andreas Alfons

See Also

`incMean`

Examples

```
data(eusilc)
weightedMean(eusilc$eqIncome, eusilc$rb050)
```

`weightedMedian` *Weighted median*

Description

Compute the weighted median (Eurostat definition).

Usage

```
weightedMedian(x, weights = NULL, sorted = FALSE, na.rm = FALSE)
```

Arguments

`x`	a numeric vector.
`weights`	an optional numeric vector giving the sample weights.
`sorted`	a logical indicating whether the observations in `x` are already sorted.
`na.rm`	a logical indicating whether missing values in `x` should be omitted.

Details

The implementation strictly follows the Eurostat definition.

Value

The weighted median of values in `x` is returned.

Author(s)

Andreas Alfons and Matthias Templ

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`arpt`, `incMedian`, `weightedQuantile`

Examples

```
data(eusilc)
weightedMedian(eusilc$eqIncome, eusilc$rb050)
```

`weightedQuantile` *Weighted quantiles*

Description

Compute weighted quantiles (Eurostat definition).

Usage

```
weightedQuantile(x, weights = NULL, probs = seq(0, 1, 0.25),
                 sorted = FALSE, na.rm = FALSE)
```

Arguments

`x`	a numeric vector.
`weights`	an optional numeric vector giving the sample weights.
`probs`	numeric vector of probabilities with values in $[0, 1]$.
`sorted`	a logical indicating whether the observations in `x` are already sorted.
`na.rm`	a logical indicating whether missing values in `x` should be omitted.

Details

The implementation strictly follows the Eurostat definition.

Value

A numeric vector containing the weighted quantiles of values in `x` at probabilities `probs` is returned. Unlike `quantile`, this returns an unnamed vector.

Author(s)

Andreas Alfons and Matthias Templ

References

Working group on Statistics on Income and Living Conditions (2004) Common cross-sectional EU indicators based on EU-SILC; the gender pay gap. *EU-SILC 131-rev/04*, Eurostat.

See Also

`incQuintile`, `weightedMedian`

Examples

```
data(eusilc)
weightedQuantile(eusilc$eqIncome, eusilc$rb050)
```

Index

∗Topic **datasets**
eusilc, 15
∗Topic **hplot**
meanExcessPlot, 27
paretoQPlot, 30
∗Topic **manip**
fitPareto, 16
minAMSE, 28
paretoScale, 31
paretoTail, 32
replaceTail, 37
reweightOut, 38
shrinkOut, 42
thetaHill, 43
thetaISE, 44
thetaLS, 46
thetaMoment, 47
thetaPDC, 48
thetaQQ, 50
thetaTM, 51
thetaWML, 53
∗Topic **package**
laeken-package, 2
∗Topic **survey**
arpr, 4
arpt, 6
bootVar, 7
calibVars, 9
calibWeights, 10
eqInc, 12
eqSS, 13
gini, 18
gpg, 20
incMean, 23
incMedian, 24
incQuintile, 25
qsr, 35
rmpg, 40
utils, 54
variance, 56
weightedMean, 58
weightedMedian, 59
weightedQuantile, 60

arpr, 4, *7–9*, *56–58*
arpt, *5*, 6, *6*, *25*, *41*, *60*

boot, *8*
boot.ci, *8*
bootVar, 7, *11*, *57, 58*

calib, *11*
calibVars, 9, *10, 11*, *39*
calibWeights, *9*, 10, *10*

eqInc, 12, *14*
eqSS, *12*, 13, *13*
eusilc, 15

fitPareto, 16, *34*, *44, 45*, *47–49*, *51, 52*, *54*

gini, *8, 9*, 18, *22*, *36*, *56–58*
ginv, *11*
gpg, 20, *56*

identify, *27, 28*, *30, 31*
incMean, 23, *59*
incMedian, *7*, 24, *60*
incQuintile, 25, *36*, *61*
is.arpr *(utils)*, 54
is.gini *(utils)*, 54
is.gpg *(utils)*, 54
is.indicator *(utils)*, 54
is.qsr *(utils)*, 54
is.rmpg *(utils)*, 54

laeken *(laeken-package)*, 2
laeken-package, 2

mean, *59*

meanExcessPlot, 27, *31, 32*
minAMSE, 28, *28*, *31, 32*, *44*

optimize, *44, 45*, *49*

paretoQPlot, *28*, 30, *32*
paretoScale, *28*, 31, *31*
paretoTail, *18*, 32, *37–39*, *42*, *44, 45*, *47–49*, *51, 52*, *54*
plot.default, *27*, *30*
print.arpr *(utils)*, 54
print.default, *29*
print.indicator *(utils)*, 54
print.minAMSE *(minAMSE)*, 28
print.rmpg *(utils)*, 54

qsr, *8, 9*, *20*, *22*, *26*, 35, *56–58*
quantile, *60*

replaceOut, *33, 34*, *39*, *42*
replaceOut *(replaceTail)*, 37
replaceTail, *18*, *34*, 37, *39*, *42*
reweightOut, *33, 34*, 38, *38*, *42*
rmpg, *8, 9*, 40, *56–58*

shrinkOut, *34*, *38, 39*, 42
subset.arpr *(utils)*, 54
subset.indicator *(utils)*, 54
subset.rmpg *(utils)*, 54

thetaHill, *18*, *29*, *34*, 43, *45*, *49*
thetaISE, *18*, *34*, 44, *44*, *49*
thetaLS, *18*, *34*, 46
thetaMoment, *18*, *34*, 47
thetaPDC, *17, 18*, *33, 34*, *44, 45*, 48
thetaQQ, *18*, *34*, 50
thetaTM, *18*, *34*, 51
thetaWML, *18*, *34*, *44*, 53

uniroot, *53*
utils, 54

variance, *5, 6*, *9*, *19–22*, *35, 36*, *40, 41*, 56

weighted.mean, *59*
weightedMean, *24*, 58
weightedMedian, *7*, *25*, 59, *61*
weightedQuantile, *26*, 60, *60*

Package 'GB2'

January 5, 2011

Version 1.0

Date 2010-12-30

Title Generalized Beta Distribution of the Second Kind: properties, likelihood, estimation.

Author Monique Graf <monique.graf@bfs.admin.ch>, Desislava Nedyalkova <desislava.nedyalkova@bfs.admin.ch>.

Maintainer Desislava Nedyalkova <desislava.nedyalkova@bfs.admin.ch>

Description GB2 is a simple package that calculates the basic properties of the Generalized Beta distribution of the second kind - density, distribution function, quantiles, moments. Functions for the full loglikelihood, the profile loglikelihood and the scores are provided. Formulae for various Laeken indicators under the GB2 are implemented. It performs maximum likelihood estimation and non-linear least squares eastimation of the model parameters. It provides various polots for the vizualization and analysis of the results.

Imports hypergeo, laeken, numDeriv, stats

Suggests simFrame, survey

License GPL (>=2)

R topics documented:

Contindic 2
Contprof 3
gb2 4
Gini 5
Indicators 6
LogDensity 8
LogLikelihood 9
MLfitGB2 10
MLfullGB2 11
MLprofGB2 13
Moments 15
NonlinearFit 16
PlotsML 19
ProfLogLikelihood 20
Thomae 21
Varest 22

Index **27**

`Contindic` *Sensitivity Analysis of Laeken Indicators on GB2 Parameters*

Description

Produces a contour plot of an indicator.

Usage

```
contindic.gb2(resol, a, p1, p2, q1, q2, fn, tit, table=FALSE)
```

Arguments

`resol`	a scalar; number of grid points horizontally and vertically.
`a`	scalar; positive parameter
`p1, p2, q1, q2`	scalars; `p1` and `q1` (`p2` and `q2`) are, respectively, the `min` and `max` values of the positive parameters of the Beta distribution `p` and `q`.
`fn`	character; the name of the function to be used for the calculation of the values to be plotted.
`tit`	string; title of the plot.
`table`	boolean; if `TRUE`, a table containing the values of the function `fn` at the different grid points is printed.

Details

An indicator is defined as a function of three parameters. The shape parameter, `a`, is held fixed.

Value

A contour plot of a given indicator for a fixed value of the shape parameter `a`.

Author(s)

Monique Graf

See Also

`contour` (package `graphics`) for more details on contour plots.

Examples

```
par(mfrow=c(2,2))
p1 <- 0.3
q1 <- 0.36
p2 <- 1.5
q2 <- 1.5
a1 <- 2.7
a2 <- 9.2
resol <- 11
rangea <- round(seq(a1,a2,length.out=4),digits=1)
```

```
arpr <- function(a,p,q) 100*arpr.gb2(0.6,a,p,q)
fonc <- "arpr"
for (a in rangea){
contindic.gb2(resol,a,p1,p2,q1,q2,arpr,"At-risk-of-poverty rate",table=TRUE)
}
```

`Contprof` *Contour Plot of the Profile Log-likelihood of the GB2 Distribution*

Description

Produces a contour plot of the profile log-likelihood, which is a function of two parameters only.

Usage

```
contprof.gb2(z, w=1, resol, low=0.1, high=20)
```

Arguments

`z`	a numeric vector; in general, the income values.
`w`	a numeric vector of the same length as `z`; the sampling weights. If not available, the function should be called only with its first argument (the weights are set to 1).
`resol`	a scalar; number of grid points horizontally and vertically. For better graph quality, we recommend a value of 100.
`low, high`	scalar; lower and upper factors for scale.

Details

The matrix containing the values to be plotted (NAs are allowed) is of size `resol` × `resol`. The locations of the grid lines at which the values of the profile log-likelihood are measured are equally-spaced values between `low` and `high` multiplied by the initial parameters.

Value

A contour plot of the profile log-likelihood. The initial Fisk estimate is added as point "F".

Author(s)

Monique Graf

See Also

`fisk` for the Fisk estimate, `ProfLogLikelihood` for the profile log-likelihood and `contour` (package `graphics`) for more details on contour plots.

gb2 *The Generalized Beta Distribution of the Second Kind*

Description

Density, distribution function, quantile function and random generation for the Generalized beta distribution of the second kind with parameters a, b, p and q.

Usage

```
dgb2(x, shape1, scale, shape2, shape3)
pgb2(x, shape1, scale, shape2, shape3)
qgb2(prob, shape1, scale, shape2, shape3)
rgb2(n, shape1, scale, shape2, shape3)
```

Arguments

x	vector of quantiles.
shape1	positive parameter.
scale	positive parameter.
shape2, shape3	positive parameters of the Beta distribution.
prob	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.

Details

The Generalized Beta distribution of the second kind with parameters shape1 $= a$, scale $= b$, shape2 $= p$ and shape3 $= q$ has density

$$f(x) = \frac{a(x/b)^{ap-1}}{bB(p,q)(1+(x/b)^a)^{p+q}}$$

for $a > 0$, $b > 0$, $p > 0$ and $q > 0$, where $B(p, q)$ is the Beta function (beta). If Z follows a Beta distribution with parameters p and q and

$$y = \frac{z}{1-z}$$

, then

$$x = b * y^{\frac{1}{a}}$$

follows the GB2 distribution.

Value

dgb2 gives the density, pgb2 the distribution function, qgb2 the quantile function, and rgb2 generates random deviates.

Author(s)

Monique Graf

References

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*, chapter 6. Wiley, Ney York.

McDonald, J. B. (1984) Some generalized functions for the size distribution of income. *Econometrica*, **52**, 647–663.

See Also

`beta` for the Beta function and `dbeta` for the Beta distribution.

Examples

```
a <- 3.9
b <- 18873
p <- 0.97
q <- 1.03
x <- qgb2(0.6, a, b, p, q)
y <- dgb2(x, a, b, p, q)
```

`Gini` *Computation of the Gini Coefficient for the GB2 Distribution and its Particular Cases.*

Description

Computes the Gini coefficient for the GB2 distribution using the function `gb2.gini`. Computes the Gini coeficient for the Beta Distribution of the Second Kind, Dagum and Singh-Maddala distributions.

Usage

```
gini.gb2(shape1, shape2, shape3)
gini.b2(shape2, shape3)
gini.dag(shape1, shape2)
gini.sm(shape1, shape3)
```

Arguments

`shape1`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.

Value

The Gini coefficient.

Author(s)

Monique Graf

References

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*, chapter 6. Wiley, Ney York.

McDonald, J. B. (1984) Some generalized functions for the size distribution of income. *Econometrica*, **52**, 647–663.

See Also

`gb2.gini`

`Indicators` *Monetary Laeken Indicators under the GB2*

Description

Functions to calculate four primary social welfare indicators under the GB2, i.e. the at-risk-of-poverty threshold, the at-risk-of-poverty rate, the relative median at-risk-of-poverty gap, and the income quintile share ratio.

Usage

```
arpt.gb2(prop, shape1, scale, shape2, shape3)
arpr.gb2(prop, shape1, shape2, shape3)
rmpg.gb2(arpr, shape1, shape2, shape3)
qsr.gb2(shape1, shape2, shape3)
main.gb2(prop, shape1, scale, shape2, shape3)
main2.gb2(prop, shape1, scale, shape12, shape13)
```

Arguments

`prop`	proportion (in general is set to 0.6).
`arpr`	the value of the at-risk-of-poverty rate.
`shape1`	positive parameter.
`scale`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.
`shape12`	the product of the two parameters `shape1` and `shape2`.
`shape13`	the product of the two parameters `shape1` and `shape3`.

Details

In June 2006, the Social Protection Committee, which is a group of officials of the European Commisiion, adopts a set of common indicators for the social protection and social inclusion process. It consists of a portfolio of 14 overarching indicators (+11 context indicators) meant to reflect the overarching objectives (a) "social cohesion" and (b) "interaction with the Lisbon strategy for growth and jobs (launched in 2000) objectives"; and of three strand portfolios for social inclusion, pensions, and health and long-term care.

The at-risk-of-poverty threshold (or ARPT) is defined as 60% of the median national equivalized income.

The at-risk-of-poverty rate (or ARPR) is defined as the share of persons with an equivalised disposable income below the ARPT.

The relative median at-risk-of-poverty gap (or RMPG) is defined as the difference between the median equivalised income of persons below the ARPT and the ARPT itself, expressed as a percentage of the ARPT.

The income quintile share ratio (or QSR) is defined as the ratio of total income received by the 20% of the country's population with the highest income (top quintile) to that received by the 20< income (lowest quintile).

Let $x_{0.5}$ be the median of a GB2 with parameters `shape1` $= a$, `scale` $= b$, `shape2` $= p$ and `shape3` $= q$. Then,

$$ARPT(a, b, p, q) = 0.6x_{0.5}$$

The ARPR being scale-free, b can be chosen arbitrarily and can be fixed to 1.

The QSR is calculated with the help of the incomplete moments of order 1.

`main.gb2` and `main2.gb2` return a vector containing the following set of GB2 indicators: the median, the mean, the ARPR, the RMPG, the QSR and the Gini coefficient. The only difference is in the input parameters.

Value

`arpt.gb2` gives the ARPT, `arpr.gb2` the ARPR, `rmpg.gb2` the RMPG, and `qsr.gb2` calculates the QSR. `main.gb2` returns a vector containing the median of the distribution, the mean of the distribution, the ARPR, the RMPG, the QSR and the Gini coefficient. `main2.gb2` produces the same output as `main.gb2`.

Author(s)

Monique Graf

References

http://ec.europa.eu/employment_social/spsi/docs/social_inclusion/2008/indicators_update2008_en.pdf

See Also

`qgb2`, `incompl.gb2`

Examples

```
a <- 3.9
b <- 18873
p <- 0.97
q <- 1.03
ap <- a*p
aq <- a*q

arpt <- arpt.gb2(0.6, a, b, p, q)
arpr <- arpr.gb2(0.6, a, p, q)
rmpg <- rmpg.gb2(arpr, a, p, q)
qsr <- qsr.gb2(a, p, q)

ind1 <- main.gb2(0.6, a, b, p, q)
ind2 <- main2.gb2(0.6, a, b, ap, aq)
```

`LogDensity` *Log Density of the GB2 Distribution*

Description

Calculates the log density of the GB2 distribution for a single value or a vector of values. Calculates the first- and second-order partial derivatives of the log density evaluated at a single value.

Usage

```
logf.gb2(x, shape1, scale, shape2, shape3)
dlogf.gb2(xi, shape1, scale, shape2, shape3)
d2logf.gb2(xi, shape1, scale, shape2, shape3)
```

Arguments

`xi`	a scalar.
`x`	a scalar or a numeric vector.
`shape1`	positive parameter.
`scale`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.

Details

We calculate $log(f(x, \theta))$, where f is the GB2 density with parameters `shape1` $= a$, `scale` $= b$, `shape2` $= p$ and `shape3` $= q$, θ is the parameter vector. We calculate the first- and second-order partial derivatives of $log(f(x, \theta))$ with respect to the parameter vector θ.

Value

Depending on the input `logf.gb2` gives the log density for a single value or a vector of values. `dlogf.gb2` gives the vector of the four first-order partial derivatives of the log density and `d2logf.gb2` gives the 4×4 matrix of second-order partial derivatives of the log density.

Author(s)

Desislava Nedyalkova

References

Brazauskas, V. (2002) Fisher information matrix for the Feller-Pareto distribution. *Statistics \& Probability Letters*, **59**, 159–167.

`LogLikelihood` *Full Log-likelihood of the GB2 Distribution*

Description

Calculates the log-likelihood, the score functions of the log-likelihood and the Fisher information matrix based on all four parameters of the GB2 distribution.

Usage

```
loglp.gb2(x, shape1, scale, shape2, shape3, w=1)
loglh.gb2(x, shape1, scale, shape2, shape3, w=1, hs)
scoresp.gb2(x, shape1, scale, shape2, shape3, w=1)
scoresh.gb2(x, shape1, scale, shape2, shape3, w=1, hs)
info.gb2(shape1, scale, shape2, shape3)
```

Arguments

`x`	vector of data values.
`shape1`	positive parameter.
`scale`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.
`w`	vector of weights.
`hs`	vector of household sizes.

Details

We express the log-likelihood as a weighted mean of $log(f)$, evaluated at the data points, where f is the GB2 density with parameters `shape1` $= a$, `scale` $= b$, `shape2` $= p$ and `shape3` $= q$. If the weights are not available, then we suppose that `w` $= 1$. `loglp.gb2` calculates the log-likelihood in the case where the data is a sample of persons and `loglh.gb2` is adapted for a sample of households. Idem for the scores, which are obtained as weighted sums of the first derivatives of $log(f)$ with respect to the GB2 parameters, evaluated at the data points. The Fisher information matrix of the GB2 was obtained by Brazauskas (2002) and is expressed in terms of the second derivatives of the log-likelihood with respect to the parameters of the GB2.

Author(s)

Monique Graf

References

Brazauskas, V. (2002) Fisher information matrix for the Feller-Pareto distribution. *Statistics \& Probability Letters*, **59**, 159–167.

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*, chapter 6. Wiley, Ney York.

MLfitGB2 *Fitting the GB2 by the Method of Maximum Likelihood Estimation*

Description

Performs maximum likelihood estimation through the general-purpose optimisation function `optim` from package `stats`.

Usage

```
main.emp(z, w=1)
mlfit.gb2(z, w=1)
```

Arguments

`z`	a numeric vector; in general, the income values.
`w`	a numeric vector of the same length as `z`; the sampling weights. If not available, the function should be called only with its first argument (the weights are set to 1).

Details

The function makes a call to `ml.gb2` and `profml.gb2`. Estimates of the GB2 parameters are calculated . Also the primary Laeken indicators of the fitted parameters and the empirical estimates of the indicators (see package `laeken`) are calculated.

Value

A list containing three different objects. The first is a data frame with the values of the fitted parameters for the full log-likelihood and the profile log-likelihood, the values of the two likelihoods, the values of the estimated indicators under the GB2 and the values of the empirical estimates of the indicators. The second and third objects are, respectively, the fit using the full log-likelihood and the fit using the profile log-likelihhod.

Author(s)

Monique Graf and Desislava Nedyalkova

See Also

`optim`, `ml.gb2`, `profml.gb2`

Examples

```
# An example of using the function mlfit.gb2
# See also the examples of ml.gb2 and mlprof.gb2

## Not run:
library(laeken)
data(eusilc)

# Income
```

```
inc <- as.vector(eusilc$eqIncome)

# Weights
w <- eusilc$rb050

# Data set
d <- data.frame(inc, w)
d <- d[!is.na(d$inc),]

# Truncate at 0
inc <- d$inc[d$inc > 0]
w   <- d$w[d$inc > 0]

# ML fit
m1 <- mlfit.gb2(inc,w)

# Results
m1[[1]]
# The fit using the full log-likelihood
m1[[2]]
# The fit using the profile log-likelihood
m1[[3]]

# ML fit, when no weights are avalable
m2 <- mlfit.gb2(inc)
# Results
m2[[1]]

## End(Not run)
```

`MLfullGB2` *Maximum Likelihood Estimation of the GB2 Based on the Full Log-likelihood*

Description

Two methods of optimization are considered: BFGS and L-BFGS-B (see `optim` documentation for more details). Initial values of the parameters to be optimized over (a, b, p and q) are given from the Fisk distribution, which is a GB2 distribution with $p = q = 1$. If m and v denote, respectively, the mean and variance of $log(z)$, then $\hat{a} = \pi/\sqrt{3 * v}$ and $\hat{b} = \exp(m)$. The function to be maximized by `optim` returns the negative of the full log-likelihood and the gradient is equal to the negative of the scores, respectively for the case of a sample of persons and a sample of households.

Usage

```
fisk(z, w=1)
fiskh(z, w=1, hs)
ml.gb2(z, w=1, method=1)
mlh.gb2(z, w=1, hs, method=1)
```

Arguments

`z`	vector of data values.

w	corresponding weights.
hs	vector of household sizes.
method	the method to be used for optim. By default, the used method is BFGS. If method = 2, methods BFGS and L-BFGS-B are used.

Details

Function fisk first calculates the mean and variance of $log(z)$ and returns the initial values of a and p under the Fisk distribution. Function fiskh first calculates the mean and variance of $log(z)$, assuming a sample of households, and returns the initial values of a and p under the Fisk distribution. Function ml.gb2 performs maximum likelihood estimation through the general-purpose optimization function optim from package stats, based on the full log-likelihood calculated in a sample of persons. Function mlh.gb2 performs maximum likelihood estimation through the general-purpose optimization function optim from package stats, based on the full log-likelihood calculated in a sample of households.

Value

A list with 1 or 2 arguments: opt1 output of BFGS fit and opt2 output of L-BFGS fit. Further values are given by the values of optim.

Author(s)

Monique Graf

References

Graf, M. and Nedyalkova, D. (2010) The Generalized Beta Distribution of the Second Kind as an Income Distribution: Properties, Likelihood and Estimation. *working paper*, SFSO.

See Also

optim for the general-purpose optimization

Examples

```
library(laeken)
data(eusilc)

# Income
inc <- as.vector(eusilc$eqIncome)

# Weights
w <- eusilc$rb050

# Data set
d <- data.frame(inc, w)
d <- d[!is.na(d$inc),]

# Truncate at 0
inc <- d$inc[d$inc > 0]
w   <- d$w[d$inc > 0]

# Fit using the full log-likelihood
fitf <- ml.gb2(inc, w)$opt1
```

```
# Fitted GB2 parameters
af <- fitf$par[1]
bf <- fitf$par[2]
pf <- fitf$par[3]
qf <- fitf$par[4]

# Likelihood
flik <- fitf$value

# If we want to compare the indicators
## Not run:
# GB2 indicators
indicf <- round(main.gb2(0.6,af,bf,pf,qf), digits=3)
# Empirical indicators
indice <- round(main.emp(inc,w), digits=3)

## End(Not run)
# Plots
## Not run: plotsML.gb2(inc,af,bf,pf,qf,w)
```

`MLprofGB2` *Maximum Likelihood Estimation of the GB2 Based on the Profile Log-likelihood*

Description

`profml.gb2` performs maximum likelihood estimation based on the profile log-likelihood through the general-purpose optimization function `optim` from package `stats`.

Usage

```
profml.gb2(z, w=1, method=1)
```

Arguments

`z`	vector of data values.
`w`	corresponding weights.
`method`	the method to be used for `optim`. By default, the used method is BFGS. If `method = 2`, methods BFGS and L-BFGS-B are used.

Details

Two methods are considered: BFGS and L-BFGS-B (see `optim` documentation for more details). Initial values of the parameters to be optimized over (a and b) are given from the Fisk distribution. The function to be maximized by `optim` is the negative of the profile log-likelihood (`proflogl.gb2`) and the gradient is equal to the negative of the scores (`profscores.gb2`).

Value

A list with 1 or 2 arguments: `opt1` output of BFGS fit and `opt2` output of L-BFGS fit. Further values are given by the values of `optim`.

Author(s)

Monique Graf

References

Graf, M. and Nedyalkova, D. (2010) The Generalized Beta Distribution of the Second Kind as an Income Distribution: Properties, Likelihood and Estimation. *working paper*, SFSO.

See Also

`optim` for the general-purpose optimization and `link{ml.gb2}` for the full log-likelihood.

Examples

```
library(laeken)
data(eusilc)

# Income
inc <- as.vector(eusilc$eqIncome)

# Weights
w <- eusilc$rb050

# Data set
d <- data.frame(inc,w)
d <- d[!is.na(d$inc),]

# Truncate at 0
inc <- d$inc[d$inc > 0]
w   <- d$w[d$inc > 0]

# Fit using the profile log-likelihood
fitp <- profml.gb2(inc, w)$opt1

# Fitted GB2 parameters
ap <- fitp$par[1]
bp <- fitp$par[2]
pp <- prof.gb2(inc, ap, bp, w)[3]
qp <- prof.gb2(inc, ap, bp, w)[4]

# Profile log-likelihood
proflik <- fitp$value

# If we want to compare the indicators
## Not run:
# GB2 indicators
indicp <- round(main.gb2(0.6,ap,bp,pp,qp), digits=3)
# Empirical indicators
indice <- round(main.emp(inc,w), digits=3)

## End(Not run)

# Plots
## Not run: plotsML.gb2(inc,ap,bp,pp,qp,w)
```

`Moments` *Moments and Other Properties of a GB2 Random Variable*

Description

These functions calculate the moments of order `k` and incomplete moments of order `k` of a GB2 random variable X as well as the expectation, the variance, the kurtosis and the skewness of $log(X)$.

Usage

```
moment.gb2(k, shape1, scale, shape2, shape3)
incompl.gb2(x, k, shape1, scale, shape2, shape3)
el.gb2(shape1, scale, shape2, shape3)
vl.gb2(shape1, shape2, shape3)
sl.gb2(shape2, shape3)
kl.gb2(shape2, shape3)
```

Arguments

`x`	vector of quantiles.
`k`	numeric; order of the moment.
`shape1`	positive parameter.
`scale`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.

Details

Let X be a random variable following a GB2 distribution with parameters `shape1` $= a$, `scale` $= b$, `shape2` $= p$ and `shape3` $= q$. Moments and incomplete moments of X exist only for $-ap \leq k \leq aq$. Moments are given by

$$E(X^k) = b^k \frac{\Gamma(p + k/a)\Gamma(q - k/a)}{\Gamma(p)\Gamma(q)}$$

This expression, when considered a function of `k`, can be viewed as the moment-generating function of $Y = log(X)$. Thus, it is useful to compute the moments of $log(X)$, which are needed for deriving, for instance, the Fisher information matrix of the GB2 distribution. Moments of $log(X)$ exist for all `k`.

Value

`moment.gb2` gives the moment of order `k`, `incompl.gb2` gives the the incomplete moment of order `k`, `El.gb2` gives the expectation of $log(X)$, `vl.gb2` gives the variance of $log(X)$, `sl.gb2` gives the skewness of $log(X)$, `kl.gb2` gives the kurtosis of $log(X)$.

Author(s)

Monique Graf

References

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*, chapter 6. Wiley, Ney York.

See Also

`gamma` for the Gamma function and related functions (`digamma`, `trigamma` and `psigamma`).

Examples

```
a <- 3.9
b <- 18873
p <- 0.97
q <- 1.03
k <- 2
x <- qgb2(0.6, a, b, p, q)
moment.gb2(k, a, b, p, q)
incompl.gb2(x, k, a, b, p, q)
vl.gb2(a, p, q)
kl.gb2(p, q)
```

`NonlinearFit` *Fitting the GB2 by Minimizing the Distance Between a Set of Indicators and Their GB2 Expressions*

Description

Fitting the parameters of the GB2 distribution by optimizing the squared weighted distance between a set of empirical indicators and the GB2 indicators using nonlinear least squares (function `nls` from package `stats`).

Usage

```
nlsfit.gb2(z, w=1)
nlsfit2.gb2(z, w=1, a.fit, b.fit, p.fit, q.fit, cva.fit)
```

Arguments

`z`	a numeric vector; in general, the income values.
`w`	a numeric vector of the same length as `z`; the sampling weights. If not available, the function should be called only with its first argument (the weights are set to 1).
`a.fit, b.fit, p.fit, q.fit`	fitted parameters, e.g. my maximum likelihood estimation, of the GB2 distribution.
`cva.fit`	coefficient of variation of the fitted parameter $a.fit$.

Details

We consider the following set of indicators:

$$A = (median, ARPR, RMPG, QSR, Gini)$$

and their corresponding GB2 expressions $A_{GB2}(a, b, p, q)$. The nonlinear model that is passed to `nls` in the function `nlsfit.gb2` is given by:

$$\sum_{i=1}^{5} c_i \left(A_{empir,i} - A_{GB2,i}(a, b, p, q)\right)^2,$$

where the weights c_i take the differing scales into account. Initial values for the parameters are taken from the Fisk distribution. Estimates of the GB2 parameters are calculated by using the generic function `coef` on the fitted model object. The second function, `mod_nlsfit.gb2`, fits the parameters of the GB2 in two consecutive steps. In the first step, we use the set of indicators, excluding the median, and their corresponding expressions in function of `a`, `ap` and `aq`. The lower and upper bounds for the algorithm "port" of `nls` are defined. Starting values are the ML estimates of the GB2 parameters `a, p` and `q`. The bounds for `a` are defined in function of the coefficient of variation of the fitted parameter `a.fit.` `ap` and `aq` are bounded so that the constraints for the existence of the moments of the GB2 distribution and the excess for calculating the Gini coefficient are fulfilled, i.e. $ap \geq 1$ and $aq \geq 2$. In the second step, only the the parameter `b` is estimated, optimizing the weighted square difference between the empirical median and the GB2 median in function of the already obtained NLS parameters `a, p` and `q`.

Value

`nlsfit.gb2` returns a list of two values: 1) a data frame containing the values of the fitted GB2 parameters, the values of the estimated GB2 indicators and the values of the empirical estimates of the indicators and 2) the fitted object. `nlsfit2.gb2` returns a list of three values: 1) a data frame containing the values of the fitted GB2 parameters, the values of the estimated GB2 indicators and the values of the empirical estimates of the indicators, 2) the first fitted object and 3) the second fitted object.

Author(s)

Monique Graf and Desislava Nedyalkova

See Also

`nls`, `Thomae`, `moment.gb2`

Examples

```
# Takes long time to run, as it makes a call to the function ml.gb2
## Not run:
library(laeken)
data(eusilc)

# Income
inc <- as.vector(eusilc$eqIncome)

# Weights
w <- eusilc$rb050
```

```
# Household size
hs <- eusilc$hsize

# Data set
d <- data.frame(inc, w, hs)
d <- d[!is.na(d$inc),]

# Truncate at 0
d <- d[d$inc > 0,]
inc <- d$inc
w   <- d$w

# ML fit, full log-likelihood
fitf <- ml.gb2(inc, w)$opt1

# Estimated parameters
af <- fitf$par[1]
bf <- fitf$par[2]
pf <- fitf$par[3]
qf <- fitf$par[4]
apf <- af*pf
aqf <- af*qf
gb2.par <- c(af, bf, pf, qf)

# GB2 indicators
indicm <- round(main.gb2(0.6,af,bf,pf,qf), digits=3)

# Empirical indicators
indice <- round(main.emp(inc,w), digits=3)

# NLS fit 1
n1 <- nlsfit.gb2(inc,w)
n1[[1]]

# Scores (partial derivatives of the log-likelihood with respect to the GB2 parameters)
scores <- matrix(nrow=length(inc), ncol=4)
for (i in 1:length(inc)){
scores[i,] <- dlogf.gb2(inc[i], af ,bf, pf, qf)
}

# Data on households only
dh <- unique(d)
hinc <- dh$inc
hw <- dh$w
hs <- dh$hs

# Estimated variance-covariance matrix of af, bf, pf and qf (EVCM)
VSC <- varscore.gb2(hinc,af,bf,pf,qf,hw,hs)
VCMP <- vepar.gb2(hinc,VSC,af,bf,pf,qf,hw,hs)[[1]]

# Standard errors of af, bf, ...
se.par <- rep(0,4)
for (i in 1:4){
se.par[i] <- sqrt(VCMP[i,i])
}

# Coefficients of variation of the fitted parameters
```

```
cv.par <- se.par/gb2.par
cvaf <- cv.par[1]

# NLS fit 2
n2 <- nlsfit2.gb2(inc, w, af, bf, pf, qf, cvaf)
n2[[1]]

## End(Not run)
```

`PlotsML` *Cumulative Distribution Plot and Kernel Density Plot for the Fitted GB2*

Description

Function `plotsML.gb2` produces two plots. The first is a plot of the empirical cumulative distribution function versus the fitted cumulative distibution function. The second is a plot of the kernel density versus the fitted GB2 density. Function `saveplot` saves locally the produced plot.

Usage

```
plotsML.gb2(z, shape1, scale, shape2, shape3, w=1)
saveplot(name, pathout)
```

Arguments

`z`	a numeric vector; in general, a vector of income values.
`shape1`	positive parameter.
`scale`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.
`w`	a numeric vector of the same length as `z`; the sampling weights. If not available, the function should be called only with its first argument (the weights are set to 1).
`name`	a character string specifying the name of the plot.
`pathout`	a character string specifying the path of the folder or device where the plot will be saved.

Details

The used kernel is "Epanechnikov" (see `plot.density`).

Author(s)

Monique Graf and Desislava Nedyalkova

`ProfLogLikelihood` *Profile Log-likelihood of the GB2 Distribution*

Description

Estimation of the parameters `shape2` $= p$ and `shape3` $= q$ of the GB2 distribution as functions of `shape1` $= a$ and `scale` $= b$, profile log-likelihood of the GB2 distribution, scores of the profile log-likelihood.

Usage

```
prof.gb2(x, shape1, scale, w=1)
proflogl.gb2(x, shape1, scale, w=1)
profscores.gb2(x, shape1, scale, w=1)
```

Arguments

`x`	vector of data values.
`shape1`	positive parameter.
`scale`	positive parameter.
`w`	vector of weights.

Details

Using the full log-likelihood equations for the GB2 distribution, the parameters p and q can be estimated as functions of a and b. These functions are plugged into the log-likelihood expression, which becomes a function of a and b only. This is obtained by reparametrizing the GB2, i.e. we set $r = \frac{p}{p+q}$ and $s = p + q$. More details can be found in Graf (2009).

Value

`prof` returns a vector containing the estimates of r, s, p, q as well as two other parameters used in the calculation of the profile log-likelihood and the scores. `proflogl.gb2` returns the value of the profile log-likelihood and `profscores.gb2` returns the vector of the first derivatives of the profile log-likelihhod with respect to a and b.

Author(s)

Monique Graf and Desislava Nedyalkova

References

Graf, M. (2009) The Log-Likelihood of the Generalized Beta Distribution of the Second Kind. *working paper*, SFSO.

`Thomae` *Maximum Excess Representation of a Generalized Hypergeometric Function Using Thomae's Theorem*

Description

Defines Thomae's arguments from the upper(`U`) and lower(`L`) parameters of a ${}_3F_2(U, L; 1)$. Computes the optimal combination leading to the maximum excess. Computes the optimal combination of Thomae's arguments and calculates the optimal representation of the ${}_3F_2(U, L; 1)$ using the `genhypergeo_series` function from package `hypergeo`. Computes the Gini coefficient for the GB2 distribution, using Thomae's theorem.

Usage

```
ULg(U, L)
combiopt(g)
Thomae(U, L, lB, tol, maxiter, debug)
gb2.gini(shape1, shape2, shape3, tol=1e-08, maxiter=10000, debug=FALSE)
```

Arguments

`U`	vector of length 3 giving the upper arguments of the generalized hypergeometric function ${}_3F_2$.
`L`	vector of length 2 giving the lower arguments of the generalized hypergeometric function ${}_3F_2$.
`g`	vector of Thomae's permuting arguments.
`lB`	ratio of beta functions (a common factor in the expression of the Gini coefficient under the GB2).
`shape1`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.
`tol`	tolerance with default 0, meaning to iterate until additional terms do not change the partial sum.
`maxiter`	maximum number of iterations to perform.
`debug`	boolean; if `TRUE`, returns the list of changes to the partial sum.

Details

More details can be found in Graf (2009).

Value

`ULg` returns a list containing Thomae's arguments and the excess, `combiopt` gives the optimal combination of Thomae's arguments (for internal use only), `Thomae` returns the optimal representation of the ${}_3F_2(U, L; 1)$, `GB2.Gini` returns the Gini coefficient under the GB2.

Author(s)

Monique Graf

References

Graf, M. (2009) An Efficient Algorithm for the Computation of the Gini coefficient of the Generalized Beta Distribution of the Second Kind. *ASA Proceedings of the Joint Statistical Meetings*, 4835–4843. American Statistical Association (Alexandria, VA).

In Proceedings of JSM 2009.

McDonald, J. B. (1984) Some generalized functions for the size distribution of income. *Econometrica*, **52**, 647–663.

See Also

`genhypergeo_series`

`Varest` *Variance Estimation of the Parameters of the GB2 Distribution*

Description

Calculation of variance estimates of the estimated GB2 parameters and the estimated GB2 indicators under cluster sampling.

Usage

```
varscore.gb2(x, shape1, scale, shape2, shape3, w=1, hs)
vepar.gb2(x, Vsc, shape1, scale, shape2, shape3, w=1, hs)
derivind.gb2(shape1, scale, shape2, shape3)
veind.gb2(x, Vpar, shape1, scale, shape2, shape3, w=1, hs)
```

Arguments

`x`	vector of data values.
`shape1`	positive parameter.
`scale`	positive parameter.
`shape2, shape3`	positive parameters of the Beta distribution.
`w`	vector of weights.
`hs`	household size.
`Vsc`	a squared matrix of size the number of parameters, e.g. the estimated design variance-covariance matrix of the estimated parameters.
`Vpar`	a squared matrix of size the number of parameters. The sandwich variance estimator of the vector of parameters.

Details

We express the log-likelihood as a weighted sum of $log(f)$, evaluated at the data points, where f is the GB2 density with parameters `shape1` $= a$, `scale` $= b$, `shape2` $= p$ and `shape3` $= q$. Knowing the first and second derivatives of $log(f)$, and using the sandwich variance estimator (see Freedman (2006)), the calculation of the variance estimates of the GB2 parameters is straightforward. We know that the GB2 estimates of the Laeken indicators are functions of the GB2 parameters. In this case, the variance estimates of the fitted indicators are obtained using the delta method. More details can be found in Graf and Nedyalkova (2010).

Value

`varscore.gb2` calculates the middle term of the sandwich variance estimator under simple random cluster sampling. `vepar.gb2` returns a list of two elements: the estimated variance-covariance matrix of the estimated GB2 parameters and the second partial derivative of the pseudo log-likelihood function. The function `veind.gb2` returns the estimated variance-covariance matrix of the estimated GB2 indicators. `derivind.gb2` calculates the numerical derivatives of the GB2 indicators and is for internal use only.

Author(s)

Monique Graf and Desislava Nedyalkova

References

Davison, A. (2003), *Statistical Models*. Cambridge University Press.

Freedman, D. A. (2006), On The So-Called "Huber Sandwich Estimator" and "Robust Standard Errors". *The American Statistician*, **60**, 299–302.

Graf, M. and Nedyalkova, D. (2010), Variance Estimation of the Maximum Likelihood Estimates of the Parameters of the GB2 Distribution of the Second Kind and of the Laeken Indicators under Cluster Sampling. Working paper, SFSO.

Pfeffermann, D. and Sverchkov, M. Yu. (2003), Fitting Generalized Linear Models under Informative Sampling. In, Skinner, C.J. and Chambers, R.L. (eds.). *Analysis of Survey Data*, chapter 12, 175–195. Wiley, New York.

Examples

```
# An example of variance estimation of the GB2 parameters,
# using the dataset "eusilcP" from the R package simFrame.
# Takes long time to run

## Not run:
library(survey)
library(simFrame)
data(eusilcP)

# Draw a sample from eusilcP
# 1-stage simple random cluster sampling of size 6000 (cluster = household)
# directly,
#s <- draw(eusilcP[, c("hid", "hsize", "eqIncome")], grouping = "hid", size = 6000)

# or setting up 250 samples, and drawing the first one.
# This sample setup can be used for running a simulation.
set.seed(12345)
scs <- setup(eusilcP, grouping = "hid", size = 6000, k = 250)
s <- draw(eusilcP[, c("region", "hid", "hsize", "eqIncome")], scs, i=1)

# The number of observations (persons) in eusilcP (58654 persons)
\dontrun{N <- dim(eusilcP)[1]}
# The number of households in eusilcP (25000 households)
Nh <- length(unique(eusilcP$hid))

# Survey design corresponding to the drawn sample
sdo = svydesign(id=~hid, fpc=rep(Nh,nrow(s)), data=s)
\dontrun{summary(sdo)}
```

```
# Truncated sample (truncate at 0)
s <- s[!is.na(s$eqIncome),]
str <- s[s$eqIncome > 0, ]
eqInc <- str$eqIncome
w <- str$.weight

# Designs for the truncated sample
sdotr <- subset(sdo, eqIncome >0)
sddtr = svydesign(id=~hid, strata=~region, fpc=NULL, weights=~.weight, data=str)
\dontrun{summary(sdotr)}
\dontrun{summary(sddtr)}

# Fit by maximum likelihood
fit <- ml.gb2(eqInc,w)$opt1
af <- fit$par[1]
bf <- fit$par[2]
pf <- fit$par[3]
qf <- fit$par[4]
mlik <- -fit$value

# Estimated parameters and indicators, empirical indicators
gb2.par <- round(c(af, bf, pf, qf), digits=3)
emp.ind <- main.emp(eqInc, w)
gb2.ind <- main.gb2(0.6, af, bf, pf, qf)

# Scores
scores <- matrix(nrow=length(eqInc), ncol=4)
for (i in 1:length(eqInc)){
scores[i,] <- dlogf.gb2(eqInc[i], af, bf, pf, qf)
}

# Data on households only
sh <- unique(str)
heqInc <- sh$eqIncome
hw <- sh$.weight
hhs <- sh$hsize
hs <- as.numeric(as.vector(hhs))

# Variance of the scores
VSC <- varscore.gb2(heqInc, af, bf, pf, qf, hw, hs)

# Variance of the scores using the explicit designs, and package survey
DV1 <- vcov(svytotal(~scores[,1]+scores[,2]+scores[,3]+scores[,4], design=sdotr))
DV2 <- vcov(svytotal(~scores[,1]+scores[,2]+scores[,3]+scores[,4], design=sddtr))

# Estimated variance-covariance matrix of the parameteres af, bf, pf and qf
VCMP <- vepar.gb2(heqInc, VSC, af, bf, pf, qf, hw, hs)[[1]]
DVCMP1 <- vepar.gb2(heqInc, DV1, af, bf, pf, qf, hw, hs)[[1]]
DVCMP2 <- vepar.gb2(heqInc, DV2, af, bf, pf, qf, hw, hs)[[1]]

\dontrun{diag(DVCMP1)/diag(VCMP)}
\dontrun{diag(DVCMP2)/diag(VCMP)}
\dontrun{diag(DV1)/diag(VSC)}
\dontrun{diag(DV2)/diag(VSC)}

# Standard errors of af, bf, pf and qf
```

```
se.par <- rep(0,4)
for (i in 1:4){
se.par[i] <- sqrt(VCMP[i,i])
}
sed1.par <- rep(0,4)
for (i in 1:4){
sed1.par[i] <- sqrt(DVCMP1[i,i])
}
sed2.par <- rep(0,4)
for (i in 1:4){
sed2.par[i] <- sqrt(DVCMP2[i,i])
}

# Estimated variance-covariance matrix of the indicators (VCMI)
VCMI <- veind.gb2(heqInc, VCMP, af, bf, pf, qf, hw, hs)
DVCMI1 <- veind.gb2(heqInc, DVCMP1, af, bf, pf, qf, hw, hs)
DVCMI2 <- veind.gb2(heqInc, DVCMP2, af, bf, pf, qf, hw, hs)

# Standard errors and confidence intervals
inCI <- rep(0,6) # empirical indicator in CI
varest.ind <- rep(0,6)
se.ind <- rep(0,6)
lci.ind <- rep(0,6)
uci.ind <- rep(0,6)
for (i in 1:6)
{
varest.ind[i] <- VCMI[i,i]
se.ind[i] <- sqrt(varest.ind[i])
lci.ind[i] <- gb2.ind[i] - 1.96*se.ind[i]
uci.ind[i] <- gb2.ind[i] + 1.96*se.ind[i]
if (lci.ind[i] <= emp.ind[i] & emp.ind[i] <= uci.ind[i]) {inCI[i] = 1}
}

#under the sampling design sdotr
inCId1 <- rep(0,6)
varestd1.ind <- rep(0,6)
sed1.ind <- rep(0,6)
lcid1.ind <- rep(0,6)
ucid1.ind <- rep(0,6)
for (i in 1:6)
{
varestd1.ind[i] <- DVCMI1[i,i]
sed1.ind[i] <- sqrt(varestd1.ind[i])
lcid1.ind[i] <- gb2.ind[i] - 1.96*sed1.ind[i]
ucid1.ind[i] <- gb2.ind[i] + 1.96*sed1.ind[i]
if (lcid1.ind[i] <= emp.ind[i] & emp.ind[i] <= ucid1.ind[i]) {inCId1[i] = 1}
}

#under the sampling design sddtr
inCId2 <- rep(0,6)
varestd2.ind <- rep(0,6)
sed2.ind <- rep(0,6)
lcid2.ind <- rep(0,6)
ucid2.ind <- rep(0,6)
for (i in 1:6)
{
varestd2.ind[i] <- DVCMI2[i,i]
```

```
sed2.ind[i] <- sqrt(varestd2.ind[i])
lcid2.ind[i] <- gb2.ind[i] - 1.96*sed2.ind[i]
ucid2.ind[i] <- gb2.ind[i] + 1.96*sed2.ind[i]
if (lcid2.ind[i] <= emp.ind[i] & emp.ind[i] <= ucid2.ind[i]) {inCId2[i] = 1}
}

#coefficients of variation .par (parameters), .ind (indicators)
cv.par <- se.par/gb2.par
cv.ind <- se.ind/gb2.ind
cvd1.par <- sed1.par/gb2.par
cvd1.ind <- sed1.ind/gb2.ind
cvd2.par <- sed2.par/gb2.par
cvd2.ind <- sed2.ind/gb2.ind


#results
res <- data.frame(am = af, bm = bf, pm = pf, qm = qf, lik = mlik,
  median = gb2.ind[[1]], mean = gb2.ind[[2]], ARPR = gb2.ind[[3]], RMPG = gb2.ind[[4]], C
  emedian = emp.ind[[1]], emean = emp.ind[[2]], eARPR = emp.ind[[3]], eRMPG = emp.ind[[4]
  cva = cv.par[1], cvb = cv.par[2], cvp= cv.par[3], cvq = cv.par[4], cvd1a = cvd1.par[1],
  cvd2a = cvd2.par[1], cvd2b = cvd2.par[2], cvd2p= cvd2.par[3], cvd2q = cvd2.par[4],
  cvmed = cv.ind[[1]], cvmean = cv.ind[[2]], cvARPR = cv.ind[[3]], cvRMPG = cv.ind[[4]],
  cvd1med = cvd1.ind[[1]], cvd1mean = cvd1.ind[[2]], cvd1ARPR = cvd1.ind[[3]], cvd1RMPG =
  cvd2med = cvd2.ind[[1]], cvd2mean = cvd2.ind[[2]], cvd2ARPR = cvd2.ind[[3]], cvd2RMPG =

res
\dontrun{inCI}

## End(Not run)
```

Index

∗Topic **distribution**
Contindic, 1
Contprof, 3
gb2, 4
Gini, 5
Indicators, 6
LogDensity, 8
LogLikelihood, 9
MLfitGB2, 10
MLfullGB2, 11
MLprofGB2, 13
Moments, 15
NonlinearFit, 16
PlotsML, 19
ProfLogLikelihood, 20
Thomae, 21
Varest, 22

arpr.gb2 *(Indicators)*, 6
arpt.gb2 *(Indicators)*, 6

beta, *4, 5*

coef, *17*
combiopt *(Thomae)*, 21
Contindic, 1
contindic.gb2 *(Contindic)*, 1
contour, *2, 3*
Contprof, 3
contprof.gb2 *(Contprof)*, 3

d2logf.gb2 *(LogDensity)*, 8
dbeta, *5*
derivind.gb2 *(Varest)*, 22
dgb2 *(gb2)*, 4
dlogf.gb2 *(LogDensity)*, 8

el.gb2 *(Moments)*, 15

fisk, *3*
fisk *(MLfullGB2)*, 11
fiskh *(MLfullGB2)*, 11

gamma, *16*
gb2, 4
gb2.gini, *5, 6*
gb2.gini *(Thomae)*, 21
genhypergeo_series, *22*
Gini, 5
gini.b2 *(Gini)*, 5
gini.dag *(Gini)*, 5
gini.gb2 *(Gini)*, 5
gini.sm *(Gini)*, 5

incompl.gb2, *7*
incompl.gb2 *(Moments)*, 15
Indicators, 6
info.gb2 *(LogLikelihood)*, 9

kl.gb2 *(Moments)*, 15

LogDensity, 8
logf.gb2 *(LogDensity)*, 8
loglh.gb2 *(LogLikelihood)*, 9
LogLikelihood, 9
loglp.gb2 *(LogLikelihood)*, 9

main.emp *(MLfitGB2)*, 10
main.gb2 *(Indicators)*, 6
main2.gb2 *(Indicators)*, 6
ml.gb2, *10*
ml.gb2 *(MLfullGB2)*, 11
mlfit.gb2 *(MLfitGB2)*, 10
MLfitGB2, 10
MLfullGB2, 11
mlh.gb2 *(MLfullGB2)*, 11
MLprofGB2, 13
moment.gb2, *17*
moment.gb2 *(Moments)*, 15
Moments, 15

nls, *17*
nlsfit.gb2 *(NonlinearFit)*, 16
nlsfit2.gb2 *(NonlinearFit)*, 16
NonlinearFit, 16

optim, *10*, *12–14*

pgb2 *(gb2)*, 4
plot.density, *19*

PlotsML, 19
plotsML.gb2 *(PlotsML)*, 19
prof.gb2 *(ProfLogLikelihood)*, 20
proflogl.gb2, *13*
proflogl.gb2 *(ProfLogLikelihood)*, 20
ProfLogLikelihood, *3*, 20
profml.gb2, *10*
profml.gb2 *(MLprofGB2)*, 13
profscores.gb2, *13*
profscores.gb2 *(ProfLogLikelihood)*, 20

qgb2, *7*
qgb2 *(gb2)*, 4
qsr.gb2 *(Indicators)*, 6

rgb2 *(gb2)*, 4
rmpg.gb2 *(Indicators)*, 6

saveplot *(PlotsML)*, 19
scoresh.gb2 *(LogLikelihood)*, 9
scoresp.gb2 *(LogLikelihood)*, 9
sl.gb2 *(Moments)*, 15

Thomae, *17*, 21

ULg *(Thomae)*, 21

Varest, 22
varscore.gb2 *(Varest)*, 22
veind.gb2 *(Varest)*, 22
vepar.gb2 *(Varest)*, 22
vl.gb2 *(Moments)*, 15

Manual: Robust Horvitz-Thompson Estimation (RHT)

Beat Hulliger and Tobias Schoch

July 12, 2011

*Contents

msvymean 2
mer 5
msvyratio 7
tsvymean 9
EA 11
ER 14

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

```
## 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	Numercial 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 prelimiary 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 unterlying 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* insted 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

Hulliger, B. (1995): Outlier robust Horvitz-Thompson estimators, *Survey Methodology* 21 (1), pp. 79-87.

Hulliger, B. (1999): Simple and robust estimators for sampling, *Proceedings of the Survey Research Methods Section*, American Statistical Association, 1999, pp. 54-63.

See Also

`svymean`

Examples

```
## 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
## interessted 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

```
## 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 crtierion (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 inital 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 assoicated with `mer`, see `plot`.

Value

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

Author(s)

Beat Hulliger and Tobias Schoch

References

Beaumont, J.-F., and Rivest, L.-P. (2009). *Dealing with outliers in survey data.* Handbook of Statistics, Sample Surveys: Theory, Methods and Inference, Eds. D. Pfeffermann and C.R. Rao, Amsterdam:Elsevier BV. Vol. 29, Chapter 16.

Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C. (1995) A limited memory algorithm for bound constrained optimization, *SIAM J. Scientific Computing* 16, pp. 1190-1208.

Hulliger, B. (1995): Outlier robust Horvitz-Thompson estimators, *Survey Methodology* 21 (1), pp. 79–87.

Hulliger, B. (1999): Simple and robust estimators for sampling, *Proceedings of the Survey Research Methods Section*, American Statistical Association, pp. 54–63.

Nocedal, J. and Wright, S. J. (2006) *Numerical Optimization*, 2nd. ed. Springer.

Examples

```
# A simple example

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

`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

```
## 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 interations (default: 50, i.e. fully iterated estimation)
`plot`	Plot the residuals? (default: `FALSE`)
`acc`	Numercial 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

Hulliger, B. (1995): Outlier robust Horvitz-Thompson estimators, *Survey Methodology* 21 (1), pp. 79-87.

Hulliger, B. (1999): Simple and robust estimators for sampling, *Proceedings of the Survey Research Methods Section*, American Statistical Association, 1999, pp. 54-63.

See Also

svyratio

Examples

```
## 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
## interessted 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

```
## S3 method for class 'survey.design':
tsvymean(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 (i.e. `attr(object, "rob")`).

`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

Hulliger, B. (1999): Simple and robust estimators for sampling, *Proceedings of the Survey Research Methods Section*, American Statistical Association, 1999, pp. 54-63.

See Also

svymean

Examples

```
## 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 interessted 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)
```

`EA`	*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 deterministiacly). 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 compuation 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

Béguin, C., and Hulliger, B. (2004). Multivariate oulier detection in incomplete survey data: The epidemic algorithm and transformed rank correlations, *Journal of the Royal Statistical Society*, A 167(Part 2.), 275–294.

Examples

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

ER *Robust EM-algorithm ER*

Description

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

Usage

```
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
`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

Little, R. and P. Smith (1987). Editing and imputation for quantitative survey data, *Journal of the American Statistical Association*, 82, 58–68.

See Also

BEM

Examples

```
data(bushfirem)
data(bushfire.weights)
ER(bushfirem, weights=bushfire.weights,alpha=0.01,steps.output=TRUE)
```

Manual: BQSR, TQSR, and MQSR

Beat Hulliger and Tobias Schoch

July 12, 2011

*Contents

BQSR 3
MQSR 5

`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

Hulliger, B. (1995): Outlier robust Horvitz-Thompson estimators, *Survey Methodology* 21 (1), pp. 79-87.

Hulliger, B. (1999): Simple and robust estimators for sampling, *Proceedings of the Survey Research Methods Section*, American Statistical Association, 1999, pp. 54-63.

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

```
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

Hulliger, B. (1995): Outlier robust Horvitz-Thompson estimators, *Survey Methodology* 21 (1), pp. 79-87.

Hulliger, B. (1999): Simple and robust estimators for sampling, *Proceedings of the Survey Research Methods Section*, American Statistical Association, 1999, pp. 54-63.

See Also

svymean, tsvymean, msvymean, BQSR.

Examples

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

Manual: Multivariate Outlier Detection and Imputation (MODI)

Beat Hulliger and Tobias Schoch

July 12, 2011

*Contents

BEM 2
GIMCD 5
POEM 7
TRC 9
EA 11
ER 14

`BEM` *BACON-EEM Algorithm for multivariate outlier detection in incomplete multivariate survey data*

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 componentwise 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.

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 funkction `EM-normal.`

Author(s)

Cédric Béguin and Beat Hulliger

References

Béguin, 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.

Billor, N., Hadi, A.S. and Vellemann, P.F. (2000). BACON: Blocked Adaptative Computationally-efficient Outlier Nominators, *Computational Statistics and Data Analysis*, 34(3), 279–298.

Schafer J.L. (2000), *Analysis of Incomplete Multivariate Data*, Monographs on Statistics and Applied Probability 72, Chapman & Hall.

Examples

```
# 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

Béguin, 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.

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 vectore `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: `POEM.r` contains the information on POEM and `POEM.i` contains the imputed data

Author(s)

Cédric Béguin and Beat Hulliger

References

Béguin, C. and 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,Technical report, EUREDIT 2002.

Examples

```
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

Béguin, C., and Hulliger, B. (2004). Multivariate oulier detection in incomplete survey data: The epidemic algorithm and transformed rank correlations, *Journal of the Royal Statistical Society*, A 167(Part 2.), 275–294.

Examples

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

`EA`	*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 deterministiacly). 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 compuation 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

Béguin, C., and Hulliger, B. (2004). Multivariate oulier detection in incomplete survey data: The epidemic algorithm and transformed rank correlations, *Journal of the Royal Statistical Society*, A 167(Part 2.), 275–294.

Examples

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

ER *Robust EM-algorithm ER*

Description

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

Usage

```
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
`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

Little, R. and P. Smith (1987). Editing and imputation for quantitative survey data, *Journal of the American Statistical Association*, 82, 58–68.

See Also

BEM

Examples

```
data(bushfirem)
data(bushfire.weights)
ER(bushfirem, weights=bushfire.weights,alpha=0.01,steps.output=TRUE)
```

Package 'rsae'

July 12, 2011

Type Package

Title Robust Small Area Estimation

Version 0.1-2

Date 2011-07-06

Author Tobias Schoch

Maintainer Tobias Schoch <tobias.schoch@fhnw.ch>

Description Robust Small Area Estimation. Robust Basic Unit- and Area-Level Models

Suggests robustbase, nlme

License GPL (>=2) | FreeBSD

LazyLoad yes

R topics documented:

rsae-package 1
fitsaemodel 2
fitsaemodel.control 4
landsat 5
makedata 7
robpredict 9
sae-internal 10
saemodel 11

Index **13**

`rsae-package` *Robust Small Area Estimation*

Description

Computes robust basic unit- and area-level and predicts area-specific means

Details

Package:	rsae
Type:	Package
Version:	0.1-2
Date:	2011-07-06
Suggests:	robustbase, nlme
License:	GPL (>=2) | FreeBSD
LazyLoad:	yes

Implemented methods:

- maximum likelihood (as reference)
- Huber-type M-estimation
- *[S-estimation; not released, yet]*
- *[Simple and robust, minimal-iterative estimation; not released, yet]*

How to:

Data analysis with `rsae` involves the following steps:

1. prepare the data/ set up the model for estimation; see `saemodel`
2. fit the model by various (robust) methods; see `fitsaemodel`
3. (robustly) predict the random effects and the area means; see `robpredict`

Author(s)

Tobias Schoch

Maintainer: Tobias Schoch <tobias.schoch@fhnw.ch>

References

Rao, J.N.K. (2003) *Small Area Estimation*, New York: John Wiley and Sons.

Richardson, A.M. and A.H. Welsh (1995) *Robust restricted maximum likelihood in mixed linear model*, Biometrics 51, pp. 1429-1439.

Schoch, T. (2011) *The robust basic unit-level small area model. A simple and fast algorithm for large datasets*, in: Proceedings of the New Technologies and Techniques Conference (NTTS), EUROSTAT, Brussels.

Sinha, S.K. and J.N.K. Rao (2009) *Robust small area estimation*, Canadian Journal of Statistics 37, pp. 381-399.

`fitsaemodel` *Fit SAE model using various methods*

Description

`fitsaemodel` is the workhorse function. It estimates SAE models that have been set up by `saemodel` (or synthetic data generated by `makedata`) by various (robust) estimation methods.

Usage

```
fitsaemodel(method, model, ...)

## S3 method for class 'fitsaemodel'
print(x, digits=3, ...)
## S3 method for class 'fitsaemodel'
summary(object, full=FALSE, digits=3, ...)
## S3 method for class 'fitsaemodel'
coef(object, type="both", ...)
```

Arguments

`method`	character string defining the method to be used; currently, either `method="ml"` for (non-robust) maximum likelihood or `method="huberm"` for Huber-type M-estimation
`model`	a `"saemodel"` object (i.e., a SAE model; see `saemodel`)
`x`	used by the `print` method
`digits`	used by the `print` and `summary` methods; number of decimal places to be shown
`object`	an object of the class `"fitsaemodel"`; i.e., a fitted model
`full`	logical, if `full=TRUE`, the `summary` method shows all it has to show (default: `full=FALSE`)
`type`	character string use in the `coef` method; it can take one of the following possibilities: `"both"`, `"ranef"`, or `"fixef"`. The first reports both, random and fixed effects (default).
`...`	additional arguments delivered to either `fitsaemodel.control`

Details

The function `fitsaemodel` is a wrapper function that calls the algorithm associated with a particular method. Two methods are currently implemented

- maximum likelihood (`method="ml"`),
- Huber-type M-estimation (`method="huberm"`).

Maximum likelihood: The call for ML is straightforward: `fitsaemodel(method="ml", model)`, where `model` is a SAE model generated by `saemodel`. Note that ML is not a robust fitting method.

Huber-type M-estimation: The call for Huber-type M-estimaton (with Huber psi-function) is: `fitsaemodel(method="huberm", model, k)`, where `model` is a SAE model generated by `saemodel`, and `k` is the robustness tuning constant of the Huber psi-function.

If your data are supposed to be heavily contaminated (**or** if the **default algorithm did not converge**), you may initialize the `fitsaemodel` alogrithm with a high-breakdown-point estimate. The **rsae** package offers two methods to initialize the algorithm, `"lts"` and `"s"`; see below. **NOTE**, you have to install the **robustbase** package in order to use these methods. The initialization methods are called in the `fitsaemodel` device (as additional argument), using

- `init="lts"`, for fast-LTS regression form **robustbase**,
- `init="s"`, for a regression S-estimator from **robustbase**.

For more details on the methods, you are refered to the documentation of **robustbase**. In general, for small to medium datasets, both methods are equivalent. For data with more than 50,000 observations, the S-estimator is considerably faster. (If the `"ml"` does not converge, you may initialize it analogously–though, it may be rather inefficient.)

Value

An instance of the class `"fitmodel"`

Author(s)

Tobias Schoch

References

Schoch, T. (2011) *Robust Basic Unit-Level Small Area Model*, Working Paper

See Also

`fitsaemodel.control`

Examples

```
#generate the synthetic data/model
mymodel <- makedata()
#compute Huber M-estimation type estimates of the model "mymodel"
#robustness tuning constant k = 2
myfittedmodel <- fitsaemodel("huberm", mymodel, k=2)
myfittedmodel
#get a summary of the model
summary(myfittedmodel)
```

---

fitsaemodel.control

Tuning parameters of fitsaemodel

Description

This function carries global settings and parameter definitions that are used by fitsaemodel (and its derivatives). Modifications of the parameters can be delivered as additional arguments in the fitsaemodel call.

Usage

```
fitsaemodel.control(niter = 40, iter = c(200, 200),
                    acc = 1e-05, init="default", ...)
```

Arguments

`niter`	integer, defining the maximum number of outer-loop iterations (default: `niter=40`)
`iter`	integer or vector of size 2, defining the maximum loops of the inner loops (default: `iter=c(200, 200)`; element 1 refers to beta; element 2 refers to v; note that d has an implicitly defined maxiter of 100 and cannot be modified)
`acc`	scalar or vector of size 4, defining the numeric tolerance used in the termination rule of the loops (default: `acc=1e-05`; the positions of elements in the vector of size 4 are: 1=acc outer-loop; 2=acc inner-loop beta; 3=acc inner-loop v; 4=acc inner-loop d).
`init`	a character string; specifies by what method the main algorithm is initialized; by default: `init="default"`; alternatively, (and provided that the **robustbase** package is installed) one may choose a high-breakdown-point initial estimate: either "lts" (fast LTS regression) or "s" (S-estimate of regression). For datasets with more than 100,000 observations, the former is rather slow. For more details on the initializing methods see the documentation of **robustbase** (`"ltsReg"` and `"lmrob.S"`).
`...`	(will be used in the future)

Details

Caution! Modifying the default values of the parameters may result in convergence failure and/or loss of convergence speed.

Value

(an object used by the robust methods)

Author(s)

Tobias Schoch

See Also

`fitsaemodel`

`landsat` *LANDSAT data: Prediction of County Crop Areas Using Survey and Satellite Data*

Description

The `landsat` data.frame is a compilation (by Battese et al., 1988) of survey and satellite data. It consists of data on segments (primary sampling unit; 1 segement =approx= 250 hectares) under corn and soybeans for 12 counties in north-central Iowa; see Details, below.

Usage

```
data(landsat)
```

Format

A data frame with 37 observations on the following 10 variables.

`SegmentsInCounty` a numeric vector; no. of segments per county

`SegementID` a numeric vector; sample segment identifier (per county)

`HACorn` a numeric vector; hectares of corn for each sample segment (as reported in the June 1978 Enumerative Survey)

`HASoybeans` a numeric vector; hectares of soybeans for each sample segment (as reported in the June 1978 Enumerative Survey)

`PixelsCorn` a numeric vector; no. of pixels classified as corn for each sample segment (LANDSAT readings)

`PixelsSoybeans` a numeric vector; no. of pixels classified as soybeans for each sample segment (LANDSAT readings)

`MeanPixelsCorn` a numeric vector; county mean number of pixels classified as corn

`MeanPixelsSoybeans` a numeric vector; county mean number of pixels classified as soybeans

`outlier` a logical vector; flags observation no. 33 as outlier

`CountyName` a factor with levels (i.e., county names) `Cerro Gordo` `Hamilton` `Worth` `Humboldt` `Franklin` `Pocahontas` `Winnebago` `Wright` `Webster` `Hancock` `Kossuth` `Hardin`

Details

The `landsat` data is a compilation (by Battese et al., 1988) of the LANDSAT satellite data from the U.S. Department of Agriculture (USDA) and the 1978 June Enumerative Survey.

Survey data: The survey data on the areas under corn and soybeans (reported in hectares) in the 37 segments of the 12 counties (north-central Iowa) have been determined by USDA Statistical Reporting Service staff, who interviewed farm operators. A segment is about 250 hectares.

Satellite data: For the LANDSAT satellite data, information is recorded as "pixels". The USDA has been engaged in research toward transforming satellite information into good estimates of crop areas at the individual pixel and segments level. A pixel is about 0.45 hectares. The satellite (LANDSAT) readings were obtained during August and September 1978.

Data for more than one sample segment are available for several counties (i.e, unbalanced data).

Observations No. 33 has been flaged as outlier (cf., Battese et al. (1988, p. 28).

Source

The data `landsat` is from Table 1 of Battese et al. (1988, p. 29).

References

Battese, G.E, R.M. Harter, and W.A. Fuller (1988) *An Error-Components Model for Prediction of County Crop Areas Using Survey and Satellite Data*, Journal of the American Statistical Association 83, pp. 28–36.

Examples

```
data(landsat)
```

`makedata` *Synthetic data generation for the basic unit-level SAE model (incl. outlier contamination)*

Description

This function serves for synthetically generating data with area-level variation. It has been written to test several estimating methods. In addition, one may introduce contamination to the laws of the model- and/or random effects (see Details, below).

Usage

```
makedata(seed=1024, intercept=1, beta=1, n=4, g=20, areaID=NULL,
         ve=1, ve.contam=41, ve.epsilon=0, vu=1, vu.contam=41,
         vu.epsilon=0)
```

Arguments

`seed`	an integer, defining the `set.seed` (default `seed=1024`)
`intercept`	either a scalar as intercept of the fixed-effects model or `NULL` (default: `intercept=1`)
`beta`	scalar or vector defining the fixed-effect coefficients (default: `beta=1`). For each given coefficient, a vector of realizations is drawn from the standard normal distribution.
`n`	integer, defining the number of units per area in balanced-data setups (default: `n=4`)
`g`	integer, defining the number of areas (default: `g=20`)
`areaID`	by default `areaID=NULL`. If one attempts to generate synthetic unbalanced data, one may call `makedata` with a vector, the elements of which area identifiers. This vector should contain a series of (integer valued) area IDs. The number of areas is set equal to the number unique IDs; see the `rsae` Vignette for more details.
`ve`	scalar, defining the model/ residual variance
`ve.contam`	scalar, defining the model variance of the outlier part in a mixture distribution (Tuckey-Huber-type contamination model). e = (1-h)*N(0, ve) + h*N(0, ve.contam)
`ve.epsilon`	scalar, defining the relative number of outliers (i.e., epsilon or h in the contamination mixture distribution). Typically, it takes values between 0 and 0.5 (but it is not restricted to this interval)
`vu`	scalar, defining the (area-level) random-effect variance
`vu.contam`	scalar, defining the (area-level) random-effect variance of the outlier part in the contamination mixture distribution (cf., `ve.contam`)
`vu.epsilon`	scalar, defining the relative number of outliers in the contamination mixture distribution of the (area-level) random effects (cf., `ve.epsilon`)

Details

The function `makedata` generates synthetic datasets that may be used to study the behavior of different estimating methods. Let y_i denote an area-specific n_i-vector of the response variable for the areas $i = 1, ..., g$. Define a $(n_i \times p)$-matrix X_i of realizations from the std. normal distribution, $N(0, 1)$, and let β denote a p-vector of regression coefficients. Now, the y_i are drawn using the law $y_i \sim N(X_i\beta, v_e I_i + v_u J_i)$ with v_e and v_u the variances of the model error and random-effect variance, respectively, and I_i and J_i denoting the identity matrix and matrix of ones, respectively.

In addition, we allow the distribution of the model/residual and area-level random effect to be contaminated (cf., Stahel and Welsh, 1997). Notably, the laws of $e_{i,j}$ and u_i are replaced by the Tukey-Huber contamination mixture:

- $e_{i,j} \sim (1 - \epsilon^{ve})N(0, v_e) + \epsilon^{ve} N(0, v_e^{\epsilon})$,
- $u_i \sim (1 - \epsilon^{vu})N(0, v_u) + \epsilon^{vu} N(0, v_u^{\epsilon})$,

where ϵ^{ve} and ϵ^{vu} regulate the degree of contamination; v_e^{ϵ} and v_e^{ϵ} define the variance of the contamination part of the mixture distribution.

Four different contamination setups are possible:

- no contamination (i.e., `ve.epsilon=vu.epsilon=0`),
- contaminated model error (i.e., `ve.epsilon!=0` and `vu.epsilon=0`),
- contaminated random effect (i.e., `ve.epsilon=0` and `vu.epsilon!=0`),
- both are conaminated (i.e., `ve.epsilon!=0` and `vu.epsilon!=0`).

Value

Instance of the class `saemodel`.

Author(s)

Tobas Schoch

References

Stahel, W.A. and A. Welsh (1997) *Approaches to robust estimation in the simplest variance components model*, Journal of Inference and Statistical Planning 57, pp. 295-319.

Examples

```
#generate synthetic data
mymodel <- makedata()
```

`robpredict` *Robust prediction of random effects, fixed effects, and area-specific means*

Description

Robust prediction of random effects, fixed effects, and area-specific means. It can predict based on new, directly delivered, areadata.

Usage

```
robpredict(fit, k, areadata=NULL)

## S3 method for class 'meanssaemodel'
print(x, digits=4, ...)
## S3 method for class 'meanssaemodel'
plot(x, y=NULL, sort=NULL, ...)
## S3 method for class 'meanssaemodel'
residuals(object, ...)
```

Arguments

`fit`	a fitted SAE model; object of class `fitsaemodel`
`k`	robustness tuning constant (of the Huber psi-function) for robust prediction. Notice that `k` does not necessarily be the same as the `k` that has been used in `fitsaemodel`.
`areadata`	numeric matrix (typically, with area-level means); the no. of rows must be equal to the no. of areas; the no. of columns must be equal to the no. of fixed-effects coefficients (incl. intercept). By default: `areadata=NULL`, i.e., predictions are based on those data that have been used to estimate the model.
`x`	object of the class `"meanssaemodel"`; this argument is only used in the `print` method.
`digits`	integer, defining the number of decimal places to be shown in the `print` method (default: `digits=4`)
`y`	has no meaning, yet! (default: `y=NULL`; needs to included in the args list, because it is part of plot's generic arg definition)
`sort`	only used in the `plot` method; if `sort="means"`, the predicted means are ploted in ascending order (default: `sort=NULL`); similarly, with `sort="fixef"` and `sort="ranef"` the predicted means are sorted along the fixed effects or the random effects, respectively
`object`	object of the class `fitsaemodel`; a fitted model used in the `residuals` method.
`...`	not used

Details

Given the robustly estimated SAE model, one considers robustly predicting the random- and fixed effect (and the final area-specific means). The tuning constant `k` regulates the degree of robustness when predicting the random effects.

Value

Instance of the S3 class `meanssaemodel`

Author(s)

Tobias Schoch

References

Schoch, T. (2011) *The robust basic unit-level small area model. A simple and fast algorithm for large datasets*, in: Proceedings of the New Technologies and Techniques Conference (NTTS), EUROSTAT, Brussels.

Examples

```
#generate the synthetic data/model
mymodel <- makedata()
#compute Huber M-estimation type estimates of the model "mymodel"
#robustness tuning constant k = 2
myfittedmodel <- fitsaemodel("huberm", mymodel, k=2)
myfittedmodel
#get a summary of the model
summary(myfittedmodel)
#robustly predict the random effects and the area-level means.
#Here, we choose the robustness tuning constant k equal to 1.8
mypredictions <- robpredict(myfittedmodel, k=1.8)
mypredictions
#a visual display of the area-specific predictions
plot(mypredictions)
```

---

sae-internal *Internal functions*

Description

These functions are for internal use only.

Author(s)

Tobias Schoch

See Also

`rsae-package`

`saemodel` *Set up a SAE model*

Description

`saemodel` is the workhorse function to set up a model (i.e., an instance of the `"saemodel"` class). It is the starting point of every model fitting exercise. Once a model has been initilized/ set up, we consider estimating its parameter.

Usage

```
saemodel(formula, area, data, type = "b", na.omit = FALSE)

## S3 method for class 'saemodel'
print(x, ...)
## S3 method for class 'saemodel'
summary(object, ...)
## S3 method for class 'saemodel'
as.matrix(x, ...)
```

Arguments

`formula`	a two-sided linear formula object describing the fixed-effects part, with the response on the RHS of the ~ operator and the terms or regressors, separated by + operators, on the LHS of the formula.
`area`	a one-sided formula object. A ~ operator followed by only one single term defining the area-specific random-effect part
`data`	data.frame
`type`	either "a" or "b" refering to J.N.K. Rao's definition of model type A (area-level model) or B (unit-level model); default is `type="b"`
`na.omit`	a logical indicating whether `NA` should be removed (default is `FALSE`). Note that none of the algorithms can cope with missing values.
`x`	an object of the class `"saemodel"` (this argument is implicitly used by the `print` and `as.matrix` methods)
`object`	an object of the class `"saemodel"` (this argument is implicitly used by the `summary` method)
`...`	not used

Details

The step of setting up a SAE model is the starting point of any (robust) SAE modeling exercise. (Use the makedata to generate a synthetic dataset; see also, below). Here, we have to define the fixed-effects- and random-effects part of the model, and to tell `R` what data it shall use.

Once a model has been initilized/ set up, we consider estimating its parameter; see `fitsaemodel`.

Value

Instance of the S3 class `"saemodel"`.

12 *saemodel*

Author(s)

Tobias Schoch

References

Rao, J.N.K. (2003). *Small Area Estimation*, New York: John Wiley and Sons.

See Also

`makedata`

Index

∗Topic **datasets**
landsat, 5
.computekappa *(sae-internal)*, 10
.fitsaemodel.huberm *(sae-internal)*, 10
.initmethod *(sae-internal)*, 10

as.matrix.saemodel *(saemodel)*, 11

coef.fitsaemodel *(fitsaemodel)*, 2

fitsaemodel, *2*, 2, *3*, *5*, *11*
fitsaemodel.control, *3*, *4*, 4

landsat, 5

makedata, *2*, 7, *12*

plot.meanssaemodel *(robpredict)*, 9
print.fitsaemodel *(fitsaemodel)*, 2
print.meanssaemodel *(robpredict)*, 9
print.saemodel *(saemodel)*, 11

residuals.meanssaemodel *(robpredict)*, 9
robpredict, *2*, 9
rsae *(rsae-package)*, 1
rsae-package, *10*
rsae-package, 1

sae-internal, 10
saemodel, *2*, *3*, 11
summary.fitsaemodel *(fitsaemodel)*, 2
summary.saemodel *(saemodel)*, 11