Package 'simPop'

July 18, 2019

Type Package

Title Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information

Version 1.2.0

URL `https://github.com/statistikat/simPop`

Depends R(>= 3.0.0), lattice, vcd, data.table

Imports laeken, plyr, MASS, Rcpp (>= 0.11.0),RcppArmadillo, e1071, parallel, nnet, doParallel, foreach, colorspace, VIM, methods,party,EnvStats,fitdistrplus,ranger,wrswoR

Suggests haven, microbenchmark, stringr, testthat, surveysd

LinkingTo Rcpp,RcppArmadillo

Description Tools and methods to simulate populations for surveys based on auxiliary data. The tools include model-based methods, calibration and combinatorial optimization algorithms. The package was developed with support of the International Household Survey Network, DFID Trust Fund TF011722 and funds from the World bank.

License GPL (>= 2)

LazyLoad yes

ByteCompile TRUE

Collate '0classes.R' 'addKnownMargins.R' 'addWeights.r' 'calibPop.R' 'calibSample.R' 'calibVars.R' 'contingencyWt.R' 'correctHeap.R' 'fitGPD.R' 'getBreaks.R' 'getCat.R' 'ipu.r' 'ipu2.r' 'meanWt.R' 'printFunctions.R' 'quantileWt.R' 'sampHH.R' 'RcppExports.R' 'silcTools.R' 'silcTools2.R' 'simAnnealingDT.R' 'simCategorical.R' 'simComponents.R' 'simContinuous.R' 'simEUSILC.R' 'simGPD.R' 'simInitSpatial.R' 'simple_dis.R' 'simPop-package.R' 'simRelation.R' 'simStructure.R' 'spBwplot.R' 'spBwplotStats.R' 'spCdf.R' 'spCdfplot.R' 'spMosaic.R' 'spPredict.R' 'spSample.R' 'spTable.R' 'specifyInput.R' 'sprague.R' 'tableWt.R' 'utility.R' 'utils.R' 'whipple.R' 'dataSets.R' 'zzz.R'

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation yes

Author Matthias Templ [aut, cre],
Alexander Kowarik [aut] (<https://orcid.org/0000-0001-8598-4130>),
Bernhard Meindl [aut],
Andreas Alfons [aut],
Mathieu Ribatet [ctb],
Johannes Gussenbauer [ctb]

Maintainer Matthias Templ <matthias.templ@gmail.com>

Repository CRAN

Date/Publication 2019-07-18 06:37:39 UTC

R topics documented:

simPop-package 2
addKnownMargins 4
addWeights<- 5
calibPop 6
calibSample 9
calibVars 11
contingencyWt 12
correctHeaps 13
correctSingleHeap 15
dataObj-class 16
eusilc13puf 17
eusilcP 20
eusilcS 21
getBreaks 23
getCat 24
get_set-methods 26
ghanaS 27
ipu 28
ipu2 30
kishFactor 34
manageSimPopObj 35
quantileWt 36
sampHH 37
silcTools2 38
simCategorical 40
simComponents 43
simContinuous 45
simEUSILC 50
simInitSpatial 53
simple_dis 55
simPopObj-class 56
simRelation 57

simStructure 59
spBwplotStats 61
spCdf 62
specifyInput 63
spMosaic 65
sprague 66
spTable 67
tableWt 68
totalsRG 69
utility 70
weighted_estimators 72
whipple 74

`simPop-package` *Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information*

Description

The production of synthetic datasets has been proposed as a statistical disclosure control solution to generate public use files out of protected data, and as a tool to create "augmented datasets" to serve as input for micro-simulation models. Synthetic data have become an important instrument for *ex-ante* assessments of policies' impact. The performance and acceptability of such a tool relies heavily on the quality of the synthetic populations, i.e., on the statistical similarity between the synthetic and the true population of interest.

Details

Multiple approaches and tools have been developed to generate synthetic data. These approaches can be categorized into three main groups: synthetic reconstruction, combinatorial optimization, and model-based generation.

The package: **simPop** is a user-friendly R-package based on a modular object-oriented concept. It provides a highly optimized S4 class implementation of various methods, including calibration by iterative proportional fitting and simulated annealing, and modeling or data fusion by logistic regression.

The following applications further shows the methods and package: We firstly demonstrated the use of **simPop** by creating a synthetic population of Austria based on the European Statistics of Income and Living Conditions (Alfons et al., 2011) including the evaluation of the quality of the generated population. In this contribution, the mathematical details of functions `simStructure`, `simCategorical`, `simContinuous` and `simComponents` are given in detail. The disclosure risk of this synthetic population has been evaluated in (Templ and Alfons, 2012) using large-scale simulation studies.

Employer-employee data were created in Templ and Filzmoser (2014) whereby the structure of companies and employees are considered.

Finally, the R package **simPop** is presented in full detail in Templ et al. (2017). In this paper - the main reference to this work - all functions and the S4 class structure of the package are described in detail. For beginners, this paper might be the starting point to learn about the methods and package.

Package:	simPop
Type:	Package
Version:	1.0.0
Date:	20017-08-07
License:	GPL (>= 2)

Author(s)

Bernhard Meindl, Matthias Templ, Andreas Alfons, Alexander Kowarik,

Maintainer: Matthias Templ <matthias.templ@gmail.com>

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[1]

A. Alfons, M. Templ (2011) Simulation of close-to-reality population data for household surveys with application to EU-SILC. *Statistical Methods & Applications*, **20** (3), 383–407. doi: 10.1007/s10260-011-0163-2

M. Templ, P. Filzmoser (2014) Simulation and quality of a synthetic close-to-reality employer-employee population. Journal of Applied Statistics, **41** (5), 1053–1072. doi: 10.1080/02664763.2013.859237[2]

M. Templ, A. Alfons (2012) Disclosure Risk of Synthetic Population Data with Application in the Case of EU-SILC. In J Domingo-Ferrer, E Magkos (eds.), *Privacy in Statistical Databases*, **6344** of Lecture Notes in Computer Science, 174–186. Springer Verlag, Heidelberg. doi: 10.1007/9783-642158384_16[3]

Examples

```
## we use synthetic eusilcS survey sample data
## included in the package to simulate a population

## create the structure
data(eusilcS)
## Not run:
## approx. 20 seconds computation time
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
## in the following, nr_cpus are selected automatically
simPop <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))
simPop <- simCategorical(simPop, additional=c("pl030", "pb220a"), method="multinom", nr_cpus
simPop
class(simPop)
regModel = ~rb090+hsize+pl030+pb220a

## multinomial model with random draws
```

[1] https://doi.org/10.18637/jss.v079.i10
[2] https://doi.org/10.1080/02664763.2013.859237
[3] https://doi.org/10.1007/978-3-642-15838-4_16

```
eusilcM <- simContinuous(simPop, additional="netIncome",
              regModel = regModel,
              upper=200000, equidist=FALSE, nr_cpus=1)
class(eusilcM)

## End(Not run)

## this is already a basic synthetic population, but
## many other functions in the package might now
## be used for fine-tuning, adding further variables,
## evaluating the quality, adding finer geographical details,
## using different methods, calibrating surveys or populations, etc.
## -- see Templ et al. (2017) for more details.
```

`addKnownMargins` *add known margins/totals*

Description

add known margins/totals for a combination of variables for the population to an object of class `simPopObj`.

Usage

```
addKnownMargins(inp, margins)
```

Arguments

`inp`	a `simPopObj` containing population and household survey data as well as optionally margins in standardized format.
`margins`	a `data.frame` containing for a combination of unique variable levels for n-variables the number of known occurences in the population. The numbers must be listed in the last column of data.frame 'margins' while the characteristics must be listed in the first 'n' columns.

Details

The function takes a data.frame containing known marginals/totals for a some variables that must exist in the population (stored in slot 'pop' of input object 'inp') and updates slot 'table' of the input object. This slot finally contains the known totals.

households are drawn from the data and new ID's are generated for the new households.

Value

an object of class `simPopObj` with updated slot 'table'.

Author(s)

Bernhard Meindl

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[4]

Examples

```
data(eusilcS)
data(eusilcP)
## Not run:
## approx. 20 seconds computation time
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
inp <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))
inp <- simCategorical(inp, additional=c("pl030", "pb220a"), method="multinom",nr_cpus=1)

margins <- as.data.frame(
  xtabs(rep(1, nrow(eusilcP)) ~ eusilcP$region + eusilcP$gender + eusilcP$citizenship))
colnames(margins) <- c("db040", "rb090", "pb220a", "freq")
inp <- addKnownMargins(inp, margins)
str(inp)

## End(Not run)
```

---

addWeights<- *Methods for function* addWeights

Description

allows to modify sampling weights of an `dataObj` or `simPopObj`-object. As input the output of `calibSample` must be used.

Usage

```
addWeights(object) <- value

## S4 replacement method for signature 'dataObj'
addWeights(object) <- value

## S4 replacement method for signature 'simPopObj'
addWeights(object) <- value
```

[4] https://doi.org/10.18637/jss.v079.i10

Arguments

object	an object of class dataObj or simPopObj.
value	a numeric vector of suitable length

Examples

```
data(eusilcS)
data(totalsRG)
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
## Not run:
## approx. 20 seconds ...
addWeights(inp) <- calibSample(inp, totalsRG)

## End(Not run)
```

---

calibPop *Calibration of 0/1 weights by Simulated Annealing*

Description

A Simulated Annealing Algorithm for calibration of synthetic population data available in a simPopObj-object. The aims is to find, given a population, a combination of different households which optimally satisfy, in the sense of an acceptable error, a given table of specific known marginals. The known marginals are also already available in slot 'table' of the input object 'inp'.

Usage

```
calibPop(inp, split, temp = 1, eps.factor = 0.05, maxiter = 200,
  temp.cooldown = 0.9, factor.cooldown = 0.85, min.temp = 10^-3,
  nr_cpus = NULL, sizefactor = 2, memory = TRUE,
  choose.temp = TRUE, choose.temp.factor = 0.2, scale.redraw = 0.5,
  observe.times = 50, observe.break = 0.05, verbose = FALSE)
```

Arguments

inp	an object of class simPopObj with slot 'table' being non-null! (see addKnownMargins).
split	given strata in which the problem will be split. Has to correspond to a column population data (slot 'pop' of input argument 'inp') . For example split = c("region"), problem will be split for different regions. Parallel computing is performed automatically, if possible.
temp	starting temperatur for simulated annealing algorithm
eps.factor	a factor (between 0 and 1) specifying the acceptance error. For example eps.factor = 0.05 results in an acceptance error for the objective function of 0.05*sum(totals).
maxiter	maximum iterations during a temperature step.

temp.cooldown
: a factor (between 0 and 1) specifying the rate at which temperature will be reduced in each step.

factor.cooldown
: a factor (between 0 and 1) specifying the rate at which the number of permutations of housholds, in each iteration, will be reduced in each step.

min.temp
: minimal temperature at which the algorithm will stop.

nr_cpus
: if specified, an integer number defining the number of cpus that should be used for parallel processing.

sizefactor
: the factor for inflating the population before applying 0/1 weights

memory
: if TRUE simulated annealing is applied in less memory intensive way. Is especially usefull if factor or population is large. For this option simulated annealing is not entirely implemented in C++, therefore it might be slower than option memory=FALSE.

choose.temp
: if TRUE temp will be rescaled according to eps and choose.temp.factor. eps is defined by the product between eps_factore and the sum over the target population margins, see addKnownMargins. Only used if memory=TRUE.

choose.temp.factor
: number between (0,1) for rescaling temp for simulated annealing. temp redefined bymax(temp,eps*choose.temp.factor). Can be usefull if simulated annealing is split into subgroups with considerably different population sizes. Only used if choose.temp=TRUE and memory=TRUE.

scale.redraw
: Only used if memory=TRUE. Number between (0,1) scaling the number of households that need to be drawn and discarded in each iteration step. The number of individuals currently selected through simulated annealing is substracted from the sum over the target population margins added to inp via addKnownMargins. This difference is divided by the median household size resulting in an estimated number of housholds that the current synthetic population differs from the population margins (~redraw_gap). The next iteration will then adjust the number of housholds to be drawn or discarded (redraw) according to max(ceiling(redraw-redraw_gap*scale.redraw),1) or max(ceiling(redraw+redraw_gap*scale.redraw),1) respectively. This keeps the number of individuals in the synthetic population relatively stable regarding the population margins. Otherwise the synthetic population might be considerably larger or smaller then the population margins, through selection of many large or small households.

observe.times
: Only used if memory=TRUE. Number of times the new value of the objective function is saved. If observe.times=0 values are not saved.

observe.break
: Only used if memory=TRUE. When objective value has been saved observe.times-times the coefficient of variation is calculated over saved values; if the coefficient of variation falls below observe.break simmulated annealing terminates. This repeats for each new set of observe.times new values of the objecive function. Can help save run time if objective value does not improve much. Disable this termination by either setting observe.times=0 or observe.break=0.

verbose — boolean variable; if TRUE some additional verbose output is provided, however only if split is NULL. Otherwise the computation is performed in parallel and no useful output can be provided.

Details

Calibrates data using simulated annealing. The algorithm searches for a (near) optimal combination of different households, by swaping housholds at random in each iteration of each temperature level. During the algorithm as well as for the output the optimal (or so far best) combination will be indicated by a logical vector containg only 0s (not inculded) and 1s (included in optimal selection). The objective function for simulated annealing is defined by the sum of absolute differences between target marginals and synthetic marginals (=marginals of synthetic dataset). The sum of target marginals can at most be as large as the sum of target marginals. For every factor-level in "split", data must at least contain as many entries of this kind as target marginals.

Possible donors are automatically generated within the procedure.

The number of cpus are selected automatically in the following manner. The number of cpus is equal the number of strata. However, if the number of cpus is less than the number of strata, the number of cpus - 1 is used by default. This should be the best strategy, but the user can also overwrite this decision.

Value

Returns an object of class simPopObj with an updated population listed in slot 'pop'.

Author(s)

Bernhard Meindl, Johannes Gussenbauer and Matthias Templ

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[5]

Examples

```
data(eusilcS) # load sample data
data(eusilcP) # population data
## Not run:
## approx. 20 seconds computation time
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
simPop <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))
simPop <- simCategorical(simPop, additional=c("pl030", "pb220a"), method="multinom", nr_cpus

# add margins
margins <- as.data.frame(
  xtabs(rep(1, nrow(eusilcP)) ~ eusilcP$region + eusilcP$gender + eusilcP$citizenship))
colnames(margins) <- c("db040", "rb090", "pb220a", "freq")
```

[5] https://doi.org/10.18637/jss.v079.i10

```
simPop <- addKnownMargins(simPop, margins)
simPop_adj2 <- calibPop(simPop, split="db040", temp=1, eps.factor=0.1,memory=TRUE)

## End(Not run)
# apply simulated annealing
## Not run:
## long computation time
simPop_adj <- calibPop(simPop, split="db040", temp=1, eps.factor=0.1,memory=FALSE)

## End(Not run)
```

`calibSample` *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.

Details

The methods return a list containing both the *g*-weights (slot `g_weights`) as well as the final weights (slot `final_weights`) (initial sampling weights adjusted by the *g*-weights.

Methods

The function provides methods with the following signatures.

Argument 'inp' must be an object of class `data.frame`, `dataObj` or `simPopObj` and the totals must be specified in either objects of class `table` or `data.frame`. If argument 'totals' is a data.frame it must be provided in a way that in the first columns n-columns the combinations of variables are listed. In the last column, the frequency counts must be specified. Furthermore, variable names of all but the last column must be available also from the sample data specified in argument 'inp'. If argument 'total' is a table (e.g. created with function `tableWt`, it must be made sure that the dimnames match the variable names (and levels) of the specified input data set.

Note

list("signature(inp=\"df_or_dataObj_or_simPopObj\", totals=\"dataFrame_or_Table\",...)") 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 and Bernhard Meindl

References

Deville, J.-C. and Saerndal, C.-E. (1992) Calibration estimators in survey sampling. *Journal of the American Statistical Association*, **87**(418), 376–382. Deville, J.-C., Saerndal, C.-E. and Sautory, O. (1993) Generalized raking procedures in survey sampling. *Journal of the American Statistical Association*, **88**(423), 1013–1020.

Examples

```
data(eusilcS)
eusilcS$agecut <- cut(eusilcS$age, 7)
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09

## for simplicity, we are using population data directly from the sample, but you get the id
totals1 <- tableWt(eusilcS[, c("agecut","rb090")], weights=eusilcS$rb050)
totals2 <- tableWt(eusilcS[, c("rb090","agecut")], weights=eusilcS$rb050)
totals3 <- tableWt(eusilcS[, c("rb090","agecut","db040")], weights=eusilcS$rb050)
totals4 <- tableWt(eusilcS[, c("agecut","db040","rb090")], weights=eusilcS$rb050)

weights1 <- calibSample(inp, totals1)
totals1.df <- as.data.frame(totals1)
weights1.df <- calibSample(inp, totals1.df)
identical(weights1, weights1.df)

# we can also use a data.frame and an optional weight vector as input
df <- as.data.frame(inp@data)
w <- inp@data[[inp@weight]]
weights1.x <- calibSample(df, totals1.df, w=inp@data[[inp@weight]])
identical(weights1, weights1.x)

weights2 <- calibSample(inp, totals2)
totals2.df <- as.data.frame(totals2)
weights2.df <- calibSample(inp, totals2.df)
identical(weights2, weights2.df)

## Not run:
## approx 10 seconds computation time ...
weights3 <- calibSample(inp, totals3)
totals3.df <- as.data.frame(totals3)
weights3.df <- calibSample(inp, totals3.df)
identical(weights3, weights3.df)

## approx 10 seconds computation time ...
weights4 <- calibSample(inp, totals4)
totals4.df <- as.data.frame(totals4)
weights4.df <- calibSample(inp, totals4.df)
identical(weights4, weights4.df)

## End(Not run)
```

`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. The following methods are implemented:

- `calibVars.default(x)`
- `calibVars.matrix(x)`
- `calibVars.matrix(x)`
- `calibVars.data.frame(x)`

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)

Bernhard Meindl and Andreas Alfons

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10

See Also

`calibSample`

Examples

```
data(eusilcS)
# default method
aux <- calibVars(eusilcS$rb090)
head(aux)
# data.frame method
aux <- calibVars(eusilcS[, c("db040", "rb090")])
head(aux)
```

`contingencyWt` *Weighted contingency coefficients*

Description

Compute (weighted) pairwise contingency coefficients.

Usage

```
contingencyWt(x, ...)
```

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.
`...`	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. The following methods are implemented:

- `contingencyWt.default(x,y,weights = NULL,...)`
- `contingencyWt.matrix(x,weights = NULL,...)`
- `contingencyWt.data.frame(x,weights = NULL,...)`

Additional parameters are:

- y: a vector that can be interpreted as factor (for the default method)
- weights: an optional numeric vector containing sample weights

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)
```

---

correctHeaps *Correct age heaping*

Description

Correct for age heaping using truncated (log-)normal distributions

Usage

```
correctHeaps(x, heaps = "10year", method = "lnorm", start = 0,
  fixed = NULL)
```

Arguments

x	numeric vector
heaps	• 5year: heaps are assumed to be every 5 years (0,5,10,...) • 10year: heaps are assumed to be every 10 years (0,10,20,...)
method	a character specifying the algorithm used to correct the age heaps. Allowed values are • lnorm: drawing from a truncated log-normal distribution. The required parameters are estimated using original input data. • norm: drawing from a truncated normal distribution. The required parameters are estimated using original input data. • unif: random sampling from a (truncated) uniform distribution
start	a numeric value for the starting of the 5 or 10 year sequences (e.g. 0, 5 or 10)
fixed	numeric index vector with observation that should not be changed

Details

Age heaping can cause substantial bias in important measures and thus age heaping should be corrected.

For method "lnorm", a truncated log-normal is fit to the whole age distribution. Then for each age heap (at 0, 5, 10, 15, ...) random numbers of a truncated log-normal (with lower and upper bound) is drawn in the interval +- 2 around the heap (rounding of degree 2) using the inverse transformation method. A ratio of randomly chosen observations on an age heap are replaced by these random draws. For the ratio the age distribution is chosen, whereas on an age heap (e.g. 5) the arithmetic means of the two neighboring ages are calculated (average counts on age 4 and age 6 for age heap equals 5, for example). The ratio on, e.g. age equals 5 is then given by the count on age 5 divided by this mean This is done for any age heap at (0, 5, 10, 15, ...).

Method "norm" replace the draws from truncated log-normals to draws from truncated normals. It depends on the age distrubution (if right-skewed or not) if method "lnorm" or "norm" should be used. Many distributions with heaping problems are right-skewed.

Method "unif" draws the mentioned ratio of observations on truncated uniform distributions around the age heaps.

Repeated calls of this function mimics multiple imputation, i.e. repeating this procedure m times provides m imputed datasets that properly reflect the uncertainty from imputation.

Value

a numeric vector without age heaps

Author(s)

Matthias Templ, Bernhard Meindl, Alexander Kowarik

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10

Examples

```
## create some artificial data
age <- rlnorm(10000, meanlog=2.466869, sdlog=1.652772)
age <- round(age[age < 93])
barplot(table(age))

## artificially introduce age heaping and correct it:
# heaps every 5 years
year5 <- seq(0, max(age), 5)
age5 <- sample(c(age, age[age %in% year5]))
cc5 <- rep("darkgrey", length(unique(age)))
cc5[year5+1] <- "yellow"
barplot(table(age5), col=cc5)
barplot(table(correctHeaps(age5, heaps="5year", method="lnorm")), col=cc5)
```

```
# heaps every 10 years
year10 <- seq(0, max(age), 10)
age10 <- sample(c(age, age[age %in% year10]))
cc10 <- rep("darkgrey", length(unique(age)))
cc10[year10+1] <- "yellow"
barplot(table(age10), col=cc10)
barplot(table(correctHeaps(age10, heaps="10year", method="lnorm")), col=cc10)

# the first 5 observations should be unchanged
barplot(table(correctHeaps(age10, heaps="10year", method="lnorm", fixed=1:5)), col=cc10)
```

`correctSingleHeap` *correctSingleHeap*

Description

Correct a specific age heap in a vector containing age in years

Usage

```
correctSingleHeap(x, heap, before = 2, after = 2, method = "lnorm",
  fixed = NULL)
```

Arguments

`x`	numeric vector representing age in years (integers)
`heap`	numeric or integer vector of length 1 specifying the year for which a heap should be corrected
`before`	numeric or integer vector of length 1 specifying the number of years before the heap that may be used to correct the heap. This input will be rounded!
`after`	numeric or integer vector of length 1 specifying the number of years after the heap that may be used to correct the heap. This input will be rounded! • `5year`: heaps are assumed to be every 5 years (0,5,10,...) • `10year`: heaps are assumed to be every 10 years (0,10,20,...)
`method`	a character specifying the algorithm used to correct the age heaps. Allowed values are • `lnorm`: drawing from a truncated log-normal distribution. The required parameters are estimated using original input data. • `norm`: drawing from a truncated normal distribution. The required parameters are estimated using original input data. • `unif`: random sampling from a (truncated) uniform distribution
`fixed`	numeric index vector with observation that should not be changed

Value

a numeric vector without age heaps

Author(s)

Matthias Templ, Bernhard Meindl, Alexander Kowarik

Examples

```
## create some artificial data
age <- rlnorm(10000, meanlog=2.466869, sdlog=1.652772)
age <- round(age[age < 93])
barplot(table(age))

## artificially introduce an age heap for a specific year
## and correct it
age23 <- c(age, rep(23, length=sum(age==23)))
cc23 <- rep("darkgrey", length(unique(age)))
cc23[24] <- "yellow"
barplot(table(age23), col=cc23)
barplot(table(correctSingleHeap(age23, heap=23, before=2, after=3, method="lnorm")), col=cc2
barplot(table(correctSingleHeap(age23, heap=23, before=5, after=5, method="lnorm")), col=cc2

# the first 5 observations should be unchanged
barplot(table(correctSingleHeap(age23, heap=23, before=5, after=5, method="lnorm",
  fixed=1:5)), col=cc23)
```

---

dataObj-class *Class* "dataObj"

Description

Objects of this class contain information on a population or survey.

Objects from the Class

Objects can be created by calls of the form `new("dataObj",...)` but are usually automatically created when using `simStructure`.

Author(s)

Bernhard Meindl and Matthias Templ

See Also

`simPopObj`

Examples

```
showClass("dataObj")

## show method, generate an object of class dataObj first
data(eusilcS)
inp <- specifyInput(data=eusilcS, hhid="db030", weight="rb050", strata="db040")
## shows some basic information:
inp
```

`eusilc13puf` *Synthetic EU-SILC 2013 survey data*

Description

This data set is synthetically generated from real Austrian EU-SILC (European Union Statistics on Income and Living Conditions) data 2013.

Format

A data frame with 13513 observations on the following 62 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`).

pid personal ID

weight sampling weights

pl031 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`).

pb190 for details, see Eurostat's code book

pe040 for details, see Eurostat's code book

pl111 for details, see Eurostat's code book

pgrossIncomeCat for details, see Eurostat's code book

pgrossIncome for details, see Eurostat's code book

hgrossIncomeCat for details, see Eurostat's code book

hgrossIncome for details, see Eurostat's code book

hgrossminusCat for details, see Eurostat's code book

hgrossminus for details, see Eurostat's code book

py010g for details, see Eurostat's code book

py021g for details, see Eurostat's code book

py050g for details, see Eurostat's code book

py080g for details, see Eurostat's code book

py090g for details, see Eurostat's code book

py100g for details, see Eurostat's code book

py110g for details, see Eurostat's code book

py120g for details, see Eurostat's code book

py130g for details, see Eurostat's code book

py140g for details, see Eurostat's code book

hy040g for details, see Eurostat's code book

hy050g for details, see Eurostat's code book

hy060g for details, see Eurostat's code book

hy070g for details, see Eurostat's code book

hy080g for details, see Eurostat's code book

hy090g for details, see Eurostat's code book

hy100g for details, see Eurostat's code book

hy110g for details, see Eurostat's code book

hy120g for details, see Eurostat's code book

hy130g for details, see Eurostat's code book

hy140g for details, see Eurostat's code book

rb250 for details, see Eurostat's code book

p119000 for details, see Eurostat's code book

p038003f for details, see Eurostat's code book

p118000i for details, see Eurostat's code book

aktivi for details, see Eurostat's code book

erwintensneu for details, see Eurostat's code book

rb050 for details, see Eurostat's code book

pb040 for details, see Eurostat's code book

hb030 for details, see Eurostat's code book

px030 for details, see Eurostat's code book

rx030 for details, see Eurostat's code book

pb030 for details, see Eurostat's code book

rb030 for details, see Eurostat's code book

hx040 for details, see Eurostat's code book

pb150 for details, see Eurostat's code book

rx020 for details, see Eurostat's code book

px020 for details, see Eurostat's code book

hx050 for details, see Eurostat's code book

eqInc for details, see Eurostat's code book

hy010 for details, see Eurostat's code book

hy020 for details, see Eurostat's code book

hy022 for details, see Eurostat's code book

hy023 for details, see Eurostat's code book

Details

The data set consists of 5977 households and is used as sample data in some of the examples in package `simPop`. Note that it is included for illustrative purposes only. The sample weights do not reflect the true population sizes of Austria and its regions.

62 variables of the original survey are simulated for 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.

Author(s)

Matthias Templ

Source

This is a synthetic data set based on Austrian EU-SILC data from 2013. The original sample was provided by Statistics Austria.

References

Eurostat (2013) Description of target variables: Cross-sectional and longitudinal.

Examples

```
data(eusilc13puf)
str(eusilc13puf)
```

`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.

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)
```

---

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.

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 break points for categorizing (semi-)continuous variables*

Description

Compute break points 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, probs = NULL, strata = NULL)
```

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) break points should be equidistant or whether there should be refinements in the lower and upper tail (see "Details").
`probs`	a numeric vector of probabilities with values in $[0, 1]$ giving quantiles to be used as (positive) break points. If supplied, this is preferred over `equidist`.
`strata`	an optional vector specifying a strata variable (e.g household ids). if specified, the mean of `x` (and also of `weights` if specified) is computed within each strata before calculating the breaks.

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 break points 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 break points.

Author(s)

Andreas Alfons and Bernhard Meindl

See Also

`getCat`, `quantileWt`

Examples

```
data(eusilcS)

# semi-continuous variable, positive break points equidistant
getBreaks(eusilcS$netIncome, weights=eusilcS$rb050)

# semi-continuous variable, positive break points 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 break points.
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 break points 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)
```

get_set-methods *Extract and modify variables from population or sample data stored in an object of class* simPopObj-class.

Description

Using samp samp<- it is possible to extract or rather modify variables of the sample data within slot data in slot sample of the simPopObj-class-object. Using pop pop<- it is possible to extract or rather modify variables of the synthetic population within in slot data in slot sample of the simPopObj-class-object.

Arguments

obj	An object of class simPopObj-class
var	variable name or index for the variable in slot 'samp' of object with the slot name to be accessed. If NULL, the entire dataset (sample or population) is returned.
value	Content replacing whatever the variable in slot var in obj currently holds.

Value

Returns an object of class simPopObj-class with the appropriate replacement.

Author(s)

Bernhard Meindl

See Also

simPopObj-class,pop, pop<-, samp<-, manageSimPopObj

Examples

```
data(eusilcS)
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040",
weight="db090")
simPopObj <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))

## get/set variables in sample-object of simPopObj
head(samp(simPopObj, var="age"))
samp(simPopObj, var="newVar") <- 1
head(samp(simPopObj, var="newVar"))
## deleting is also possible
samp(simPopObj, var="newvar") <- NULL
head(samp(simPopObj, var="newvar"))
## extract multiple variables
head(samp(simPopObj, var=c("db030","db040")))
```

```
## get/set variables in pop-object of simPopObj
head(pop(simPopObj, var="age"))
pop(simPopObj, var="newVar") <- 1
head(pop(simPopObj, var="newVar"))
## deleting is also possible
pop(simPopObj, var="newvar") <- NULL
head(pop(simPopObj, var="newvar"))
## extract multiple variables
head(pop(simPopObj, var=c("db030","db040")))
```

`ghanaS` *Synthetic GLSS survey data*

Description

This data set is synthetically generated from real GLSS (Ghana Living Standards Survey) data.

Format

A data frame with 36970 observations on the following 14 variables.

hhid integer; the household ID.

hsize integer; the number of persons in the household.

region factor; the region in which the household is located (levels `western`, `central`, `greater accra`, `volta`, `eastern`, `ashanti`, `brong ahafo`, `northern`, `upper east` and `upper west`).

clust factor; the enumeration area.

age integer; the person's age.

sex factor; the person's sex (levels `male` and `female`).

relate factor; the relationship with the household head (levels `head`, `spouse`, `child`, `grandchild`, `parent/parentlaw`, `son/daughterlaw`, `other relative`, `adopted child`, `househelp` and `non_relative`).

nation factor; the person's nationality (levels `ghanaian birth`, `ghanaian naturalise`, `burkinabe`, `malian`, `nigerian`, `ivorian`, `togolese`, `liberian`, `other ecowas`, `other africa` and `other`).

ethnic factor; the person's ethnicity (levels `akan`, `all other tribes`, `ewe`, `ga-dangbe`, `grusi`, `guan`, `gurma`, `mande` and `mole-dagbani`).

religion factor; the person's religion (levels `catholic`, `anglican`, `presbyterian`, `methodist`, `pentecostal`, `spiritualist`, `other christian`, `moslem`, `traditional`, `no religion` and `other`).

highest_degree factor; the person's highest degree of education (levels `none`, `mlsc`, `bece`, `voc/comm`, `teacher trng a`, `teacher trng b`, `gce 'o' level`, `ssce`, `gce 'a' level`, `tech/prof cert`, `tech/prof dip`, `hnd`, `bachelor`, `masters`, `doctorate` and `other`).

occupation factor; the person's occupation (levels armed forces and other security personnel, clerks, craft and related trades workers, elementary occupations, legislators, senior officials and managers, none, plant and machine operators and assemblers, professionals, service workers and shop and market sales workers, skilled agricultural and fishery workers, and technicians and associate professionals).

income numeric; the person's annual income.

weight numeric; the sample weights.

Details

The data set consists of 8700 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 Ghana and its regions. The resulting population data is about 100 times smaller than the real population size to save computation time.

Only some of the variables in the original survey are included in this example data set. Furthermore, categories are aggregated for certain variables due to the large number of possible outcomes in the original survey data.

Source

This is a synthetic data set based on GLSS data from 2006. The original sample was provided by Ghana Statistical Service.

References

Ghana Statistical Service (2008) Ghana Living Standards Survey: Report of the fifth round.

Examples

```
data(ghanaS)
summary(ghanaS)
```

---

ipu *iterative proportional updating*

Description

adjust sampling weights to given totals based on household-level and/or individual level constraints

Usage

```
ipu(inp, con, hid = NULL, eps = 1e-07, verbose = FALSE)
```

Arguments

inp	a data.frame or data.table containing household ids (optionally), counts for household and/or personal level attributes that should be fitted.
con	named list with each list element holding a constraint total with list-names relating to column-names in inp.
hid	character vector specifying the variable containing household-ids within inp or NULL if such a variable does not exist.
eps	number specifiying convergence limit
verbose	if TRUE, ipu will print some progress information.

Author(s)

Bernhard Meindl

Examples

```
# basic example
inp <- as.data.frame(matrix(0, nrow=8, ncol=6))
colnames(inp) <- c("hhid","hh1","hh2","p1","p2","p3")
inp$hhid <- 1:8
inp$hh1[1:3] <- 1
inp$hh2[4:8] <- 1
inp$p1 <- c(1,1,2,1,0,1,2,1)
inp$p2 <- c(1,0,1,0,2,1,1,1)
inp$p3 <- c(1,1,0,2,1,0,2,0)
con <- list(hh1=35, hh2=65, p1=91, p2=65, p3=104)
res <- ipu(inp=inp, hid="hhid", con=con, verbose=FALSE)

# more sophisticated
# load sample and population data
data(eusilcS)
data(eusilcP)

# variable generation and preparation
eusilcS$hsize <- factor(eusilcS$hsize)

# make sure, factor levels in sample and population match
eusilcP$region <- factor(eusilcP$region, levels = levels(eusilcS$db040))
eusilcP$gender <- factor(eusilcP$gender, levels = levels(eusilcS$rb090))
eusilcP$hsize  <- factor(eusilcP$hsize , levels = levels(eusilcS$hsize))

# generate input matrix
# we want to adjust to variable "db040" (region) as household variables and
# variable "rb090" (gender) as individual information
samp <- data.table(eusilcS)
pop <-  data.table(eusilcP)
setkeyv(samp, "db030")
hh <- samp[!duplicated(samp$db030),]
hhpop <- pop[!duplicated(pop$hid),]
```

```
# reg contains for each region the number of households
reg <- data.table(model.matrix(~db040 +0, data=hh))
# hsize contains for each household size the number of households
hsize <- data.table(model.matrix(~factor(hsize) +0, data=hh))

# aggregate persons-level characteristics per household
# gender contains for each household the number of males and females
gender <- data.table(model.matrix(~db030+rb090 +0, data=samp))
setkeyv(gender, "db030")
gender <- gender[, lapply(.SD, sum), by = key(gender)]

# bind together and use it as input
inp <- cbind(reg, hsize, gender)

# the totals we want to calibrate to
con <- c(
  as.list(xtabs(rep(1, nrow(hhpop)) ~ hhpop$region)),
  as.list(xtabs(rep(1, nrow(hhpop)) ~ hhpop$hsize)),
  as.list(xtabs(rep(1, nrow(eusilcP)) ~ eusilcP$gender))
)
# we need to have the same names as in 'inp'
names(con) <- setdiff(names(inp), "db030")

# run ipu und check results
res <- ipu(inp=inp, hid="db030", con=con, verbose=TRUE)

is <- sapply(2:(ncol(res)-1), function(x) {
  sum(res[,x]*res$weights)
})
data.frame(required=unlist(con), is=is)
```

`ipu2` *Iterative Proportional Updating*

Description

Adjust sampling weights to given totals based on household-level and/or individual level constraints.

Usage

```
ipu2(dat, hid = NULL, conP = NULL, conH = NULL, epsP = 1e-06,
  epsH = 0.01, verbose = FALSE, w = NULL, bound = 4,
  maxIter = 200, meanHH = TRUE, allPthenH = TRUE, returnNA = TRUE,
  looseH = FALSE, numericalWeighting = NULL, check_hh_vars = TRUE,
  conversion_messages = FALSE)
```

Arguments

dat	a data.table containing household ids (optionally), base weights (optionally), household and/or personal level variables (numerical or categorical) that should be fitted.
hid	name of the column containing the household-ids within dat or NULL if such a variable does not exist.
conP	list or (partly) named list defining the constraints on person level. The list elements are contingency tables in array representation with dimnames corresponding to the names of the relevant calibration variables in dat. If a numerical variable is to be calibrated, the respective list element has to be named with the name of that numerical variable. Otherwise the list element shoud NOT be named.
conH	list or (partly) named list defining the constraints on household level. The list elements are contingency tables in array representation with dimnames corresponding to the names of the relevant calibration variables in dat. If a numerical variable is to be calibrated, the respective list element has to be named with the name of that numerical variable. Otherwise the list element shoud NOT be named.
epsP	numeric value or list (of numeric values and/or arrays) specifying the convergence limit(s) for conP. The list can contain numeric values and/or arrays which must appear in the same order as the corresponding constraints in conP. Also, an array must have the same dimensions and dimnames as the corresponding constraint in conP.
epsH	numeric value or list (of numeric values and/or arrays) specifying the convergence limit(s) for conH. The list can contain numeric values and/or arrays which must appear in the same order as the corresponding constraints in conH. Also, an array must have the same dimensions and dimnames as the corresponding constraint in conH.
verbose	if TRUE, some progress information will be printed.
w	name if the column containing the base weights within dat or NULL if such a variable does not exist. In the latter case, every observation in dat is assigned a starting weight of 1.
bound	numeric value specifying the multiplier for determining the weight trimming boundary if the change of the base weights should be restricted, i.e. if the weights should stay between 1/bound*w and bound*w.
maxIter	numeric value specifying the maximum number of iterations that should be performed.
meanHH	if TRUE, every person in a household is assigned the mean of the person weights corresponding to the household. If "geometric", the geometric mean is used rather than the arithmetic mean.
allPthenH	if TRUE, all the person level calibration steps are performed before the houshold level calibration steps (and meanHH, if specified). If FALSE, the houshold level calibration steps (and meanHH, if specified) are performed after everey person level calibration step. This can lead to better convergence properties in certain cases but also means that the total number of calibration steps is increased.

returnNA — if TRUE, the calibrated weight will be set to NA in case of no convergence.

looseH — if FALSE, the actual constraints conH are used for calibrating all the hh weights. If TRUE, only the weights for which the lower and upper thresholds defined by conH and epsH are exceeded are calibrated. They are however not calibrated against the actual constraints conH but against these lower and upper thresholds, i.e. conH-conH*epsH and conH+conH*epsH.

numericalWeighting — If NULL computeLinear from the pacakge survey sd will be used.

check_hh_vars — If TRUE check for non-unique values inside of a household for variables in household constraints

conversion_messages — show a message, if inputs need to be reformatted. This can be useful for speed optimizations if ipu2 is called several times with similar inputs (for example bootstrapping)

Details

This function implements the weighting procedure described here[6].

conP and conH are contingency tables, which can be created with xtabs. The dimnames of those tables should match the names and levels of the corresponding columns in dat.

maxIter, epsP and epsH are the stopping criteria. epsP and epsH describe relative tolerances in the sense that

$$1 - epsP < \frac{w_{i+1}}{w_i} < 1 + epsP$$

will be used as convergence criterium. Here i is the iteration step and wi is the weight of a specific person at step i.

The algorithm performs best if all varables occuring in the constraints (conP and conH) as well as the household variable are coded as factor-columns in dat. Otherwise, conversions will be necessary which can be monitored with the conversion_messages argument. Setting check_hh_vars to FALSE can also incease the performance of the scheme.

Value

The function will return the input data dat with the calibrated weights calibWeight as an additional column as well as attributes. If no convergence has been reached in maxIter steps, and returnNA is TRUE (the default), the column calibWeights will only consist of NAs. The attributes of the table are attributes derived from the data.table class as well as the following.

converged — Did the algorithm converge in maxIter steps?

iterations — The number of iterations performed.

conP, conH, epsP, epsH — See Arguments.

conP_adj, conH_adj — Adjusted versions of conP and conH

formP, formH — Formulas that were used to calculate conP_adj and conH_adj based on the output table.

[6] http://www.ajs.or.at/index.php/ajs/article/viewFile/doi10.17713ajs.v45i3.120/512

Author(s)

Alexander Kowarik, Gregor de Cillia

See Also

ipu

Examples

```
data(eusilcS)
setDT(eusilcS)
eusilcS <- eusilcS[, list(db030,hsize,db040,age,rb090,netIncome,db090,rb050)]

## rename columns
setnames(eusilcS, "rb090", "gender")
setnames(eusilcS, "db040", "state")
setnames(eusilcS, "db030", "household")
setnames(eusilcS, "rb050", "weight")

## some recoding
# generate age groups
eusilcS[, agegroup := cut(age, c(-Inf, 10*1:9, Inf), right = FALSE)]
# some recoding of netIncome for reasons of simplicity
eusilcS[is.na(netIncome), netIncome := 0]
eusilcS[netIncome < 0, netIncome := 0]
# set hsize to 1,...,5+
eusilcS[, hsize := cut(hsize, c(0:4, Inf), labels = c(1:4, "5+"))]
# treat households as a factor variable
eusilcS[, household := as.factor(household)]

## example for base weights assuming a simple random sample of households stratified per reg
eusilcS[, regSamp := .N, by = state]
eusilcS[, regPop := sum(weight), by = state]
eusilcS[, baseWeight := regPop/regSamp]

## constraints on person level
# age
conP1 <- xtabs(weight ~ agegroup, data = eusilcS)
# gender by region
conP2 <- xtabs(weight ~ gender + state, data = eusilcS)
# personal net income by gender
conP3 <- xtabs(weight*netIncome ~ gender, data = eusilcS)

## constraints on household level
conH1 <- xtabs(weight ~ hsize + state, data = eusilcS, subset = !duplicated(household))

# array of convergence limits for conH1
epsH1 <- conH1
epsH1[1:4,] <- 0.005
epsH1["5+",] <- 0.2

# without array epsH1
```

```
calibweights1 <- ipu2(eusilcS, hid = "household",
                      conP = list(conP1, conP2, netIncome = conP3),
                      conH = list(conH1),
                      epsP = list(1e-06, 1e-06, 1e-03),
                      epsH = 0.01,
                      bound = NULL, verbose = TRUE,  maxIter = 200)

# with array epsH1, base weights and bound
calibweights2 <- ipu2(eusilcS, hid = "household",
                      conP = list(conP1, conP2),
                      conH = list(conH1),
                      epsP = 1e-06,
                      epsH = list(epsH1),
                      w = "baseWeight",
                      bound = 4, verbose = TRUE, maxIter = 200)

# show an adjusted version of conP and the original
attr(calibweights2, "conP_adj")
attr(calibweights2, "conP")
```

`kishFactor` *Kish Factor*

Description

Compute the kish factor for a specific weight vector

Usage

```
kishFactor(w)
```

Arguments

`w`	a numeric vector with weights

Value

The function will return the the kish factor

Author(s)

Alexander Kowarik

Examples

```
kishFactor(rep(1,10))
kishFactor(rlnorm(10))
```

`manageSimPopObj` *get and set variables from population or sample data stored in an object of class* `simPopObj`.

Description

This functions allows to get or set variables in slots `pop` and `sample` of `simPopObj`-objects. This is a utility function that is useful for writing custom wrapper functions.

Usage

```
manageSimPopObj(x, var, sample = FALSE, set = FALSE, values = NULL)
```

Arguments

`x`	an object of class `simPopObj`.
`var`	character vector of length 1; variable name that should be set or extracted.
`sample`	a logical indicating whether `var` should be extracted/set from slot 'sample' (TRUE) or slot 'pop' (FALSE).
`set`	logical; if TRUE, argument 'values' is set to either the sample or population data stored in 'x', depending on argument 'sample'. If FALSE, the desired variable given by 'var' is returned from either the sample or the pop slot of 'x'.
`values`	vector; if 'set' is TRUE, then this vector is used to update the variable of sample or population data depending of choice of argument 'sample'.

Value

An object of class `simPopObj` (if 'set' is TRUE) or a vector (if 'set' is FALSE).

Author(s)

Bernhard Meindl and Matthias Templ

Examples

```
data(eusilcS)
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040",
  weight="db090")
simPopObj <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))

(manageSimPopObj(simPopObj, var="age", sample=FALSE, set=FALSE))
(manageSimPopObj(simPopObj, var="age", sample=TRUE, set=FALSE))
```

`quantileWt` *Weighted sample quantiles*

Description

Compute quantiles taking into account sample weights. The following methods are implemented:

- `quantileWt.default(x,weights=NULL,probs=seq(0,1,0.25),na.rm=TRUE,...)`
- `quantileWt.dataObj(x,vars,probs=seq(0,1,0.25),na.rm=TRUE,...)`

Additional parameters are:

- weights an optional numeric vector containing sample weights.
- vars a character vector of length 1 specifying a variable name that is available in the data-slot of `x` and which is used for the calculation.
- 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`.

Usage

```
quantileWt(x, ...)
```

Arguments

`x`	a numeric vector.
`...`	for the generic function `quantileWt` additional arguments to be passed to methods. Additional arguments not included in the definition of the methods are currently ignored.

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 and Bernhard Meindl

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))

# dataObj-method
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
(quantileWt(inp, vars="netIncome"))
```

---

sampHH *Sample households from given microdata.*

Description

The function samples households from microdata containing personal and household information.

Usage

```
sampHH(pop, sizefactor = 1, hid = "hid", strata = "region",
  hsize = NULL)
```

Arguments

pop	data frame containing households and persons
sizefactor	factor of how many times the initial population should be resampled
hid	string specifying the name of the household-id variable in the data.
strata	can be used to sample within strata.
hsize	string specifying the name of the household size variable in the data.

Details

households are drawn from the data and new ID's are generated for the new households.

Value

the data frame of new households.

Author(s)

Bernhard Meindl, Matthias Templ and Johannes Gussenbauer

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10

Examples

```
data(eusilcP)
pop <- eusilcP
colnames(pop)[3] <- "hhsize"

system.time(x1 <- sampHH(pop, strata="region", hsize="hhsize"))
dim(x1)
## Not run:
## approx. 10 second computation time ...
system.time(x1 <- sampHH(pop, sizefactor=4, strata="region", hsize="hhsize"))
dim(x1)
system.time(x2 <- sampHH(pop, strata=NULL, hsize="hhsize"))

pop <- pop[,-which(colnames(pop)=="hhsize")]
system.time(y1 <- sampHH(pop, strata="region", hsize=NULL))
system.time(y2 <- sampHH(pop, strata=NULL, hsize=NULL))

## End(Not run)
```

`silcTools2` *Utility functions for EU-SILC data*

Description

Various utility functions mainly used for simulating EU-SILC data

Usage

```
loadSILC(file = NULL, filed = NULL, filer = NULL, filep = NULL,
  fileh = NULL, year = 2013, country = "Austria")

mergeSILC(filed, filer, fileh, filep)

checkCol(x, y)

chooseSILCvars(x, vars = c("db030", "db040", "rb030", "rb080", "rb090",
  "pl031", "pb220a", "py010g", "py021g", "py050g", "py080g", "py090g",
  "py100g", "py110g", "py120g", "py130g", "py140g", "hy040g", "hy050g",
  "hy060g", "hy070g", "hy080g", "hy090g", "hy100g", "hy110g", "hy120g",
  "hy130g", "hy140g", "db090", "rb050", "pb190", "pe040", "pl051", "pl111",
  "rb010"), country = NULL)

modifySILC(x, country = "Austria")
```

Arguments

file	data set in R binary format, csv or sav (SPSS) of merged EU-SILC data.
filed	data set including the household register information
filer	data set including the personal register information
filep	data set including the personal information
fileh	data set including the household information
year	year of origin
country	country
x	public-use file (for checkCol function) or orginal data
y	scientific-use file (for checkCol function)
vars	variables to be selected for function chooseSILCvars

Details

Collection of functions to import, select and modify data EU-SILC data. Either file (merged data) or single files have to be provided for loadSILC().

Author(s)

Matthias Templ

Examples

```
## Not run:
x <- loadSILC("new_workfile.RData")
filed <- "zielvar_d_eurostat2013.sav"
filer <- "zielvar_r_eurostat2013.sav"
filep <- "zielvar_p_eurostat2013.sav"
fileh <- "zielvar_h_eurostat2013.sav"
suf4 <- loadSILC(filed = filed,
                 filer = filer,
                 filep = filep,
                 fileh = fileh)

## End(Not run)
## Not run:
filed <- "zielvar_d_eurostat2013.sav"
filer <- "zielvar_r_eurostat2013.sav"
filep <- "zielvar_p_eurostat2013.sav"
fileh <- "zielvar_h_eurostat2013.sav"
suf4 <- loadSILC(filed = filed,
                 filer = filer,
                 filep = filep,
                 fileh = fileh)
suf <- mergeSILC(d = suf4[["d"]],
                 r = suf4[["r"]],
                 h = suf4[["h"]],
                 p = suf4[["p"]])
```

```
## End(Not run)
data(eusilc13puf)
## instead of scientific-use file or
## original data we took the 2006 synthetic data
data(eusilcS)
## check which columns of y are in x
checkCol(eusilc13puf, eusilcS)
## Not run:
## on original silc data to extract needed variables for SGA project on SILC
x <- loadSILC("new_workfile.RData")
chooseSILCvars(x)

## End(Not run)
## Not run:
## wrapper to prepare SILC data
## on original silc data
x <- loadSILC("new_workfile.RData")
x <- chooseSILCvars(x)
modifySILC(x)

## End(Not run)
```

`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(simPopObj, additional, method = c("multinom",
  "distribution", "ctree", "cforest", "ranger"), limit = NULL,
  censor = NULL, maxit = 500, MaxNWts = 1500, eps = NULL,
  nr_cpus = NULL, regModel = NULL, seed = 1, verbose = FALSE,
  by = "strata")
```

Arguments

`simPopObj`	a `simPopObj` containing population and household survey data as well as optionally margins in standardized format.
`additional`	a character vector specifying additional categorical variables available in the sample object of `simPopObj` 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). "ctree" for using Classification trees "cforest" for using random forest (implementation in package party) "ranger" for using random forest (implementation in package ranger)

limit — if method is "multinom", this can be used to account for structural zeros. If only one additional variable is requested, a named list of lists should be supplied. The names of the list components specify the predictor variables for which to limit the possible outcomes of the response. For each predictor, a list containing the possible outcomes of the response for each category of the predictor can be supplied. The probabilities of other outcomes conditional on combinations that contain the specified categories of the supplied predictors are set to 0. If more than one additional variable is requested, such a list of lists can be supplied for each variable as a component of yet another list, with the component names specifying the respective variables.

censor — if method is "multinom", this can be used to account for structural zeros. If only one additional variable is requested, a named list of lists or data.frames should be supplied. The names of the list components specify the categories that should be censored. For each of these categories, a list or 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. If more than one additional variable is requested, such a list of lists or data.frames can be supplied for each variable as a component of yet another list, with the component names specifying the respective variables.

maxit, MaxNWts — control parameters to be passed to multinom and nnet. See the help file for nnet.

eps — a small positive numeric value, or NULL (the default). In the former case and if method is "multinom", estimated probabilities smaller than this are assumed to result from structural zeros and are set to exactly 0.

nr_cpus — if specified, an integer number defining the number of cpus that should be used for parallel processing.

regModel — allows to specify the variables or model that is used when simulating additional categorical variables. The following choices are available if different from NULL.

- 'basic'only the basic household variables (generated with simStructure) are used.
- 'available'all available variables (that are common in the sample and the synthetic population such as previously generated varaibles) excluding id-variables, strata variables and household sizes are used for the modelling. This parameter should be used with care because all factors are automatically used as factors internally.
- formula-objectUsers may also specify a specifiy formula (class 'formula') that will be used. Checks are performed that all required variables are available.

	If method 'distribution' is used, it is only possible to specify a vector of length one containing one of the choices described above. If parameter 'regModel' is NULL, only basic household variables are used in any case.
`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.
`verbose`	set to TRUE if additional print output should be shown.
`by`	defining which variable to use as split up variable of the estimation. Defaults to the strata variable.

Details

The number of cpus are selected automatically in the following manner. The number of cpus is equal the number of strata. However, if the number of cpus is less than the number of strata, the number of cpus - 1 is used by default. This should be the best strategy, but the user can also overwrite this decision.

Value

An object of class `simPopObj` containing survey data as well as the simulated population data including the categorical variables specified by argument `additional`.

Note

The basic household structure needs to be simulated beforehand with the function `simStructure`.

Author(s)

Bernhard Meindl, Andreas Alfons, Stefan Kraft, Alexander Kowarik, Matthias Templ

References

B. Meindl, M. Templ, A. Kowarik, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[7]

A. Alfons, M. Templ (2011) Simulation of close-to-reality population data for household surveys with application to EU-SILC. *Statistical Methods & Applications*, **20** (3), 383–407. doi: 10.1080/02664763.2013.859237[8]

See Also

`simStructure`, `simRelation`, `simContinuous`, `simComponents`

[7] https://doi.org/10.18637/jss.v079.i10
[8] https://doi.org/10.1080/02664763.2013.859237

Examples

```
data(eusilcS) # load sample data
## Not run:
## approx. 20 seconds computation time
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
## in the following, nr_cpus are selected automatically
simPop <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))
simPop <- simCategorical(simPop, additional=c("pl030", "pb220a"), method="multinom", nr_cpus
simPop

## 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(simPopObj, total = "netIncome", components = c("py010n",
  "py050n", "py090n", "py100n", "py110n", "py120n", "py130n", "py140n"),
  conditional = c(getCatName(total), "pl030"),
  replaceEmpty = c("sequential", "min"), seed)
```

Arguments

`simPopObj`	a `simPopObj`-object.
`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 conditioning 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

An object of class `simPopObj` containing survey data as well as the simulated population data including the components of the continuous variable specified by `total` and `components`.

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 and Bernhard Meindl

References

B. Meindl, M. Templ, A. Kowarik, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[9]

A. Alfons, M. Templ (2011) Simulation of close-to-reality population data for household surveys with application to EU-SILC. *Statistical Methods & Applications*, **20** (3), 383–407. doi: 10.1080/02664763.2013.859237[10]

See Also

`simStructure`, `simCategorical`, `simContinuous`, `simEUSILC`

Examples

```
data(eusilcS)
## Not run:
## approx. 20 seconds computation time
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize",
  strata="db040", weight="db090")
simPopObj <- simStructure(data=inp, method="direct",
  basicHHvars=c("age", "rb090", "hsize", "pl030", "pb220a"))
simPopObj <- simContinuous(simPopObj, additional = "netIncome",
  regModel = ~rb090+hsize+pl030+pb220a+hsize,
  method="multinom", upper=200000, equidist=FALSE, nr_cpus=1)

# categorize net income for use as conditioning variable
sIncome <- manageSimPopObj(simPopObj, var="netIncome", sample=TRUE, set=FALSE)
sWeight <- manageSimPopObj(simPopObj, var="rb050", sample=TRUE, set=FALSE)
```

[9] https://doi.org/10.18637/jss.v079.i10
[10] https://doi.org/10.1080/02664763.2013.859237

```
pIncome <- manageSimPopObj(simPopObj, var="netIncome", sample=FALSE, set=FALSE)

breaks <- getBreaks(x=unlist(sIncome), w=unlist(sWeight), upper=Inf, equidist=FALSE)
simPopObj <- manageSimPopObj(simPopObj, var="netIncomeCat", sample=TRUE,
  set=TRUE, values=getCat(x=unlist(sIncome), breaks))
simPopObj <- manageSimPopObj(simPopObj, var="netIncomeCat", sample=FALSE,
  set=TRUE, values=getCat(x=unlist(pIncome), breaks))

# simulate net income components
simPopObj <- simComponents(simPopObj=simPopObj, total="netIncome",
  components=c("py010n","py050n","py090n","py100n","py110n","py120n","py130n","py140n"),
  conditional = c("netIncomeCat", "pl030"), replaceEmpty = "sequential", seed=1 )

class(simPopObj)

## 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(simPopObj, additional = "netIncome",
  method = c("multinom", "lm", "poisson"), zeros = TRUE,
  breaks = NULL, lower = NULL, upper = NULL, equidist = TRUE,
  probs = NULL, gpd = TRUE, threshold = NULL, est = "moments",
  limit = NULL, censor = NULL, log = TRUE, const = NULL,
  alpha = 0.01, residuals = TRUE, keep = TRUE, maxit = 500,
  MaxNWts = 1500, tol = .Machine$double.eps^0.5, nr_cpus = NULL,
  eps = NULL, regModel = "basic", byHousehold = NULL,
  imputeMissings = FALSE, seed, verbose = FALSE, by = "strata")
```

Arguments

`simPopObj`	a `simPopObj` holding household survey data, population data and optionally some margins.
`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 categories, `"lm"`, for using (two-step) regression models combined with random error terms and `"poisson"` for using Poisson regression for count variables.

`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 break points for categorizing the variable specified by `additional`. If `NULL`, break points 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 break points should be equidistant or whether there should be refinements in the lower and upper tail (see `getBreaks`).

`probs` numeric vector with values in $[0, 1]$; if `method` is `"multinom"` and `breaks` is `NULL`, this gives probabilities for quantiles to be used as (positive) break points. If supplied, this is preferred over `equidist`.

`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.

`limit` an optional named list of lists; if multinomial models are computed, this can be used to account for structural zeros. The names of the list components specify the predictor variables for which to limit the possible outcomes of the response. For each predictor, a list containing the possible outcomes of the response for each category of the predictor can be supplied. The probabilities of other outcomes conditional on combinations that contain the specified categories of the supplied predictors are set to 0. Currently, this is only implemented for more than two categories in the response.

`censor` an optional named list of lists or `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 list or `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"` and `zeros` is `TRUE`, 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.

`nr_cpus` if specified, an integer number defining the number of cpus that should be used for parallel processing.

`eps` a small positive numeric value, or `NULL` (the default). In the former case and if (multinomial) log-linear models are computed, estimated probabilities smaller than this are assumed to result from structural zeros and are set to exactly 0.

`regModel` allows to specify the model that should be for the simulation of the additional continuous variable. The following choices are possible:

- 'basic'only the basic household-variables (generated with `simStructure`) are used.
- 'available'all available variables (that are common in the sample and the syntetic population (e.g. previously generated variables) are used for the modeling. Should be used with care because all variables are automatically used as factors!
- formula-object: Users may also specify a specific formula (class 'formula') that will be used. Checks are performed that all required variables are available.

byHousehold	if NULL, simulated values are used as is. If either 'sum', 'mean' or 'random' is specified, the values are aggregated and each member of the household gets the same value (mean, sum or a random value) assigned.
imputeMissings	if TRUE, missing values in variables that are used for the underlying model are imputed using hock-deck.
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.
verbose	(logical) if TRUE, additional output is written to the promt
by	defining which variable to use as split up variable of the estimation. Defaults to the strata variable.

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.

The number of cpus are selected automatically in the following manner. The number of cpus is equal the number of strata. However, if the number of cpus is less than the number of strata, the number of cpus - 1 is used by default. This should be the best strategy, but the user can also overwrite this decision.

Value

An object of class simPopObj containing survey data as well as the simulated population data including the continuous variable specified by additional and possibly simulated categories for the desired continous variable.

Note

The basic household structure and any other categorical predictors need to be simulated beforehand with the functions simStructure and simCategorical, respectively.

Author(s)

Bernhard Meindl, Andreas Alfons, Alexander Kowarik (based on code by Stefan Kraft)

References

B. Meindl, M. Templ, A. Kowarik, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[11]

A. Alfons, M. Templ (2011) Simulation of close-to-reality population data for household surveys with application to EU-SILC. *Statistical Methods & Applications*, **20** (3), 383–407. doi: 10.1080/02664763.2013.859237[12]

See Also

`simStructure`, `simCategorical`, `simComponents`, `simEUSILC`

Examples

```
data(eusilcS)
## Not run:
## approx. 20 seconds computation time
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
simPop <- simStructure(data=inp, method="direct",
  basicHHvars=c("age", "rb090", "hsize", "pl030", "pb220a"))

regModel = ~rb090+hsize+pl030+pb220a

# multinomial model with random draws
eusilcM <- simContinuous(simPop, additional="netIncome",
              regModel = regModel,
              upper=200000, equidist=FALSE, nr_cpus=1)
class(eusilcM)

## End(Not run)

## Not run:
# two-step regression
eusilcT <- simContinuous(simPop, additional="netIncome",
              regModel = "basic",
              method = "lm")
class(eusilcT)

## End(Not run)
```

[11] https://doi.org/10.18637/jss.v079.i10
[12] https://doi.org/10.1080/02664763.2013.859237

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, probs = NULL, 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 break points 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 break points for categorizing the personal income. If missing, break points 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 break points should be equidistant or whether there should be refinements in the lower and upper tail (see `getBreaks`).

`probs` numeric vector with values in $[0, 1]$; if `method` is `"multinom"` and `breaks` is `NULL`, this gives probabilities for quantiles to be used as (positive) break points. If supplied, this is preferred over `equidist`.

`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.

`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 resampling 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

An object of class `simPopObj` containing the simulated EU-SILC population data as well as the underlying sample.

Note

This is a wrapper calling `simStructure`, `simCategorical`, `simContinuous` and `simComponents`.

Author(s)

Andreas Alfons and Stefan Kraft and Bernhard Meindl

See Also

`simStructure`, `simCategorical`, `simContinuous`, `simComponents`

Examples

```
data(eusilcS) # load sample data

## Not run:
## long computation time
# 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)
```

`simInitSpatial` *Generation of smaller regions given an existing spatial variable and a table.*

Description

This function allows to manipulate an object of class `simPopObj` in a way that a new variable containing smaller regions within an already existing broader region is generated. The distribution of the smaller region within the broader region is respected.

Usage

```
simInitSpatial(simPopObj, additional, region, tspatialP = NULL,
  tspatialHH = NULL, eps = 0.05, maxIter = 100, nr_cpus = NULL,
  seed = 1, verbose = FALSE)
```

Arguments

`simPopObj`	an object of class `simPopObj`.
`additional`	a character vector of length one holding the variable name of the variable containing smaller geographical units. This variable name must be available as a column in input argument `tspatial`.
`region`	a character vector of length one holding the variable name of the broader region. This variable must be available in the input `tspatial` as well as in the sample and population slots of input `simPopObj`.
`tspatialP`	a data.frame (or data.table) containing three columns. The broader region (with the variable name being the same as in input `region`, the smaller geographical units (with the variable name being the same as in input `additional`) and a third column containing a numeric vector holding counts of persons. This argument or tspatialHH has to be provided.
`tspatialHH`	a data.frame (or data.table) containing three columns. The broader region (with the variable name being the same as in input `region`, the smaller geographical units (with the variable name being the same as in input `additional`) and a third column containing a numeric vector holding counts of households. This argument or tspatialP has to be provided.
`eps`	relative deviation of person counts if person and household counts are provided
`maxIter`	maximum number of iteration for adjustment if person and household counts are provided
`nr_cpus`	if specified, an integer number defining the number of cpus that should be used for parallel processing.
`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.
`verbose`	TRUE/FALSE if some information should be shown during the process

Details

The distributional information must be contained in an input table that holds combinations of characteristics of the broader region and the smaller regions as well as population counts (which may be available from a census).

Value

An object of class `simPopObj` with an additional variable in the synthetic population slot.

Author(s)

Bernhard Meindl and Alexander Kowarik

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[13]

Examples

```
data(eusilcS)
data(eusilcP)

# no districts are available in the population, so we have to generate those
# we randomly assign districts within "region" in the eusilc population data
# each hh has the same district
simulate_districts <- function(inp) {
  hhid <- "hid"
  region <- "region"

  a <- inp[!duplicated(inp[,hhid]),c(hhid, region)]
  spl <- split(a, a[,region])
  regions <- unique(inp[,region])

  tmpres <- lapply(1:length(spl), function(x) {
    codes <- paste(x, 1:sample(3:9,1), sep="")
    spl[[x]]$district <- sample(codes, nrow(spl[[x]]), replace=TRUE)
    spl[[x]]
  })
  tmpres <- do.call("rbind", tmpres)
  tmpres <- tmpres[,-c(2)]
  out <- merge(inp, tmpres, by.x=c(hhid), by.y=hhid, all.x=TRUE)
  invisible(out)
}

eusilcP <- data.table(simulate_districts(eusilcP))
# we generate the input table using the broad region (variable 'region')
# and the districts, we have generated before.
#Generate table with household counts by district
```

[13] https://doi.org/10.18637/jss.v079.i10

```
tabHH <- eusilcP[!duplicated(hid),.(Freq=.N),by=.(db040=region,district)]
setkey(tabHH,db040,district)
#Generate table with person counts by district
tabP <- eusilcP[,.(Freq=.N),by=.(db040=region,district)]
setkey(tabP,db040,district)

# we generate a synthetic population
setnames(eusilcP,"region","db040")
setnames(eusilcP,"hid","db030")
inp <- specifyInput(data=eusilcP, hhid="db030", hhsize="hsize", strata="db040",population=TR
simPopObj <- simStructure(data=inp, method="direct", basicHHvars=c("age", "gender"))
## Not run:
# use only HH counts
simPopObj1 <- simInitSpatial(simPopObj, additional="district", region="db040", tspatialHH=ta
tspatialP=NULL)

# use only P counts
simPopObj2 <- simInitSpatial(simPopObj, additional="district", region="db040", tspatialHH=NU
tspatialP=tabP)

# use P and HH counts
simPopObj3 <- simInitSpatial(simPopObj, additional="district", region="db040", tspatialHH=ta
tspatialP=tabP)

## End(Not run)
```

`simple_dis` *Simple generation of new variables*

Description

Fast simulation of new variables based on univariate distributions

Usage

```
univariate.dis(puf, data, additional, weights, value = "data",
  fNA = NA)

conditional.dis(puf, data, additional, conditional, weights,
  value = "data", fNA = NA)
```

Arguments

`puf`	data for which one additional column specified by function argument 'additional' is simulated
`data`	donor data
`additional`	name of variable to be simulated

weights	sampling weights from data
value	if "data" then the puf including the additional variable is returned, otherwise only the simulated vector.
fNA	only used with missing values if another code as NA should be used
conditional	conditioning variable

Details

Function uni.distribution: random draws from the weighted univariate distribution of the original data

Function conditional.dis: random draws from the weighted conditional distribution (conditioned on a factor variable)

This are simple functions to produce structural variables, variables that should have the same categories as given ones. For more advanced methods see simCategorical()

Author(s)

Lydia Spies, Matthias Templ

See Also

simCategorical

Examples

```
## we don't have original data, so let's use eusilc
data(eusilc13puf)
data(eusilcS)
v1 <- univariate.dis(eusilcS, eusilc13puf, additional = "db040",
weights = "rb050", value = "vector")
table(v1)
table(eusilc13puf$db040)
## we don't have original data, so let's use eusilc
##data(eusilc13puf)
##data(eusilcS)
##v1 <- conditional.dis(eusilcS, eusilc13puf, additional = "pb190",
##  conditional = "db040", weights = "rb050")
##table(v1) / sum(table(v1))
##table(eusilc13puf$pb190) / sum(table(eusilc13puf$pb190))
```

---

simPopObj-class *Class* "simPopObj"

Description

An object that is used throughout the package containing information on the sample (in slot sample), the population (slot pop) and optionally some margins in form of a table (slot table).

Objects from the Class

Objects are automatically created in function `simStructure`.

Author(s)

Bernhard Meindl and Matthias Templ

See Also

`dataObj`

Examples

```
showClass("simPopObj")

## show method: generate an object of class simPop first
data(eusilcS)
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
eusilcP <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))
class(eusilcP)
## shows some basic information:
eusilcP
```

---

simRelation *Simulate categorical variables of population data*

Description

Simulate categorical variables of population data taking relationships between household members into account. The household structure of the population data needs to be simulated beforehand using `simStructure`.

Usage

```
simRelation(simPopObj, relation = "relate", head = "head",
  direct = NULL, additional = c("nation", "ethnic", "religion"),
  limit = NULL, censor = NULL, maxit = 500, MaxNWts = 2000,
  eps = NULL, nr_cpus = NULL, seed)
```

Arguments

`simPopObj`	a `simPopObj` containing population and household survey data as well as optionally margins in standardized format.
`relation`	a character string specifying the columns of `dataS` and `dataP`, respectively, that define the relationships between the household members.

`head` a character string specifying the category of the variable given by `relation` that identifies the household head.

`direct` a character string specifying categories of the variable given by `relation`. Simulated individuals with those categories directly inherit the values of the additional variables from the household head. The default is `NULL` such that no individuals directly inherit value from the household head.

`additional` a character vector specifying additional categorical variables of `dataS` that should be simulated for the population data.

`limit` this can be used to account for structural zeros. If only one additional variable is requested, a named list of lists should be supplied. The names of the list components specify the predictor variables for which to limit the possible outcomes of the response. For each predictor, a list containing the possible outcomes of the response for each category of the predictor can be supplied. The probabilities of other outcomes conditional on combinations that contain the specified categories of the supplied predictors are set to 0. If more than one additional variable is requested, such a list of lists can be supplied for each variable as a component of yet another list, with the component names specifying the respective variables.

`censor` this can be used to account for structural zeros. If only one additional variable is requested, a named list of lists or `data.frame`s should be supplied. The names of the list components specify the categories that should be censored. For each of these categories, a list or `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. If more than one additional variable is requested, such a list of lists or `data.frame`s can be supplied for each variable as a component of yet another list, with the component names specifying the respective variables.

`maxit, MaxNWts`
control parameters to be passed to `multinom` and `nnet`. See the help file for `nnet`.

`eps` a small positive numeric value, or `NULL` (the default). In the former case, estimated probabilities smaller than this are assumed to result from structural zeros and are set to exactly 0.

`nr_cpus` if specified, an integer number defining the number of cpus that should be used for parallel processing.

`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

The values of a new variable are simulated in three steps, where the second step is optional. First, the values of the household heads are simulated with multinomial log-linear models. Second, individuals directly related to the corresponding household head (as specified by the argument `direct`) inherit the value of the latter. Third, the values of the remaining individuals are simulated with multinomial log-linear models in which the value of the respective household head is used as an additional predictor.

The number of cpus are selected automatically in the following manner. The number of cpus is equal the number of strata. However, if the number of cpus is less than the number of strata, the number of cpus - 1 is used by default. This should be the best strategy, but the user can also overwrite this decision.

Value

An object of class `simPopObj` containing survey data as well as 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 Bernhard Meindl

See Also

`simStructure`, `simCategorical`, `simContinuous`, `simComponents`

Examples

```
data(ghanaS) # load sample data
samp <- specifyInput(data=ghanaS, hhid="hhid", strata="region", weight="weight")
ghanaP <- simStructure(data=samp, method="direct", basicHHvars=c("age", "sex", "relate"))
class(ghanaP)

## Not run:
## long computation time ...
ghanaP <- simRelation(simPopObj=ghanaP, relation="relate", head="head")
str(ghanaP)

## End(Not run)
```

`simStructure` *Simulate the household structure of population data*

Description

Simulate basic categorical variables that define the household structure (typically variables such as household ID, age and gender) of population data by resampling from survey data.

Usage

```
simStructure(dataS, method = c("direct", "multinom", "distribution"),
  basicHHvars, seed = 1)
```

Arguments

dataS	an object of class dataObj containing household survey data that is usually generated with specifyInput.
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).
basicHHvars	a character vector specifying important variables for the household structure that need to be available in dataS. Typically variables such as age or sex may be 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

An object of class simPopObj containing the simulated population household structure as well as the underlying sample that was provided as input.

Note

The function sample is used, which gives results incompatible with those from < 2.2.0 and produces a warning the first time this happens in a session.

Author(s)

Bernhard Meindl and Andreas Alfons

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[14]

See Also

simCategorical, simContinuous, simComponents, simEUSILC

Examples

```
data(eusilcS)
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
eusilcP <- simStructure(data=inp, method="direct", basicHHvars=c("age", "rb090"))
```

[14] https://doi.org/10.18637/jss.v079.i10

```
class(eusilcP)
eusilcP
```

`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

References

A. Alfons, M. Templ (2011) Simulation of close-to-reality population data for household surveys with application to EU-SILC. *Statistical Methods & Applications*, **20** (3), 383–407. doi: 10.1007/s10260011016321[15]

See Also

`spCdfplot`, `ecdf`, `approx`

Examples

```
data(eusilcS)
cdfS <- spCdf(eusilcS$netIncome, weights = eusilcS$rb050)
plot(cdfS, type="s")
```

`specifyInput` *create an object of class 'dataObj' required for further processing*

Description

create an standardized input object of class 'dataObj' containing information on weights, household ids, household sizes, person ids and optionally strata. Outputs of this function are typically used in `simStructure`.

[15] https://doi.org/10.1007/s10260-011-0163-2

Usage

```
specifyInput(data, hhid, hhsize = NULL, pid = NULL, weight = NULL,
  strata = NULL, population = FALSE)
```

Arguments

data	a data.frame or data.table featuring sample data.
hhid	character vector of length 1 specifying variable containing household ids within slot data.
hhsize	character vector of length 1 specifying variable containing household sizes within slot data. If NULL, household sizes are automatically calculated.
pid	character vector of length 1 specifying variable containing person ids within slot data. If NULL, person ids are automatically calculated.
weight	character vector of length 1 specifying variable holding sampling weights within slot data.
strata	character vector of length 1 specifing variable name within slot data of variable holding information on strata, e.g. regions or NULL if such variable does not exist.
population	TRUE/FALSE vector of length 1 specifing if the data object is a sample or a population NULL if such variable does not exist.

Author(s)

Bernhard Meindl

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[16]

Examples

```
data(eusilcS)
inp <- specifyInput(data=eusilcS, hhid="db030", weight="rb050", strata="db040")
class(inp)
inp
```

[16] https://doi.org/10.18637/jss.v079.i10

`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, method = c("split", "color"), ...)
```

Arguments

`x`	An object of class `"spTable"` created using function `spTable`.
`method`	A character string specifying the plot method. Possible values are `"split"` to plot the expected population sizes on the left hand side and the realized population sizes on the right hand side, and `"color"`
`...`	if `method` is `"split"`, further arguments to be passed to `cotabplot`. If `method` is `"color"`, further arguments to be passed to `strucplot`

Details

If `method` is `"split"`, the two tables of expected and realized population sizes are combined into a single table, with an additional conditioning variable indicating expected and realized values. A conditional plot of this table is then produced using `cotabplot`.

Author(s)

Andreas Alfons and Bernhard Meindl

References

M. Templ, B. Meindl, A. Kowarik, A. Alfons, O. Dupriez (2017) Simulation of Synthetic Populations for Survey Data Considering Auxiliary Information. *Journal of Statistical Survey*, **79** (10), 1–38. doi: 10.18637/jss.v079.i10[17]

A. Alfons, M. Templ (2011) Simulation of close-to-reality population data for household surveys with application to EU-SILC. *Statistical Methods & Applications*, **20** (3), 383–407. doi: 10.1080/02664763.2013.859237[18]

See Also

`spTable`, `cotabplot`, `strucplot`

[17] `https://doi.org/10.18637/jss.v079.i10`
[18] `https://doi.org/10.1080/02664763.2013.859237`

Examples

```
set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
samp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize",
  strata="db040", weight="db090")
eusilcP <- simStructure(data=samp, method="direct", basicHHvars=c("age","rb090"))
abb <- c("B","LA","Vi","C","St","UA","Sa","T","Vo")
tab <- spTable(eusilcP, select=c("rb090", "db040", "hsize"))

# expected and realized population sizes
spMosaic(tab, method = "split",
  labeling=labeling_border(abbreviate=c(db040=TRUE)))

# realized population sizes colored according to relative
# differences with expected population sizes
spMosaic(tab, method = "color",
  labeling=labeling_border(abbreviate=c(db040=TRUE)))
```

---

sprague *Sprague index (multipliers)*

Description

Using the Sprague multipliers, the age counts are estimated for each year having 5-years interval data as input.

Usage

```
sprague(x)
```

Arguments

x	numeric vector of age counts in five-year intervals

Details

The input is population counts of age classes 0-4, 5-9, 10-14, ... , 77-74, 75-79, 80+.

Value

Population counts for age 0, 1, 2, 3, 4, ..., 78, 79, 80+.

Author(s)

Matthias Templ

References

G. Calot and J.-P. Sardon. Methodology for the calculation of Eurostat's demographic indicators. Detailed report by the European Demographic Observatory

http://www.cedefop.europa.eu/files/Methodology_for_the_calculation_of_Eurostats_demographic_indicators.pdf

See Also

whipple

Examples

```
## example from the world bank
x <- data.frame(age=as.factor(c(
  "0-4",
  "5-9","10-14","15-19", "20-24",
  "25-29","30-34","35-39","40-44","45-49",
  "50-54","55-59","60-64","65-69","77-74","75-79","80+"
    )),
  pop=c(1971990, 2095820,2157190, 2094110,2116580,   2003840, 1785690,
        1502990, 1214170, 796934,  627551,  530305, 488014,
        364498, 259029,158047,  125941)
)

s  <- sprague(x[,2])
s

all.equal(sum(s), sum(x[,2]))
```

---

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(inp, select)
```

Arguments

inp	an object of class simPopObj containing household survey and simulated population data.
select	character; vector defining the columns in slots 'pop' and 'sample' of argument 'input' that should be used for tabulation.

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

Sampling weights are automatically used from the input object 'inp'!

Author(s)

Andreas Alfons and Bernhard Meindl

See Also

`spMosaic`, `tableWt`

Examples

```
set.seed(1234)  # for reproducibility
data(eusilcS)   # load sample data
samp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize",
  strata="db040", weight="db090")
eusilcP <- simStructure(data=samp, method="direct", basicHHvars=c("age", "rb090"))
res <- spTable(eusilcP, select = c("age", "rb090"))
class(res)
res
```

`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")
```

---

totalsRG *Population totals Region times Gender for Austria 2006*

Description

Population characteristics Region times Gender from Austria.

Using samp samp<- it is possible to extract or rather modify variables of the sample data within slot data in slot sample of the simPopObj-class-object. Using pop pop<- it is possible to extract or rather modify variables of the synthetic population within in slot data in slot sample of the simPopObj-class-object.

Format

totalsRG: A data frame with 18 observations on the following 3 variables.

list("rb090") gender; a factor with levels `female` `male`

list("db040") region; a factor with levels `Burgenland` `Carinthia` `Lower Austria,` `Salzburg` `Styria` `Tyrol` `Upper Austria` `Vienna` `Vorarlberg`

list("Freq") totals; a numeric vector

totalsRGtab: a two-dimensional table holding the same information

Details

Population totals Region times Gender for Austria 2006

Population characteristics Region times Gender from Austria.

Source

StatCube - statistical data base, `http://www.statistik.at`

StatCube - statistical data base, `http://www.statistik.at/`

Examples

```
data(totalsRG)
totalsRG
data(totalsRGtab)
totalsRGtab


data(totalsRG)
totalsRG
data(totalsRGtab)
totalsRGtab
```

---

utility *Utility measures*

Description

Various utility measues that basically compares two data sets

Usage

```
utility(x, y, type = c("all", "compareColumns", "compareRows",
  "compareRowsHH", "compareNA"), hhid = NULL)

utilityModal(x, y, varx, vary = NULL)

utilityIndicator(x, y)
```

Arguments

x	a data.frame, typically the original data set. For utilityIndicator this should be a vector of length 1.
y	a data.frame, typically the corresponding synthetic data set. For utilityIndicator this should be a vector of length 1.
type	which measure • compareColumnscompares the intersection of variables • compareRowscompares the number of rows • compareRowsHHcompares the number of housholds • compareNAcompares the number of missings
hhid	index or name of variable containing the houshold ID
varx	name or index of a variable in data.frame x
vary	NULL or name or index of a variable in data.frame y corresponding to variable varx in data.frame x. If NULL, the names of the selected variable should be the same in both x and y.

Value

the measure(s) of interest

Functions

- utility: comparisons of two data sets
- utilityModal: comparison of number of categories
- utilityIndicator: difference between two values

Author(s)

Matthias Templ, Maxime Bergeaut

Examples

```
data(eusilcS)
data(eusilcP)
## for fast caluclations, took a subsample
eusilcP <- eusilcP[1:15000, ]
utility(eusilcS, eusilcP)
data(eusilcS)
data(eusilcP)
utilityModal(eusilcS, eusilcP, "age")
utilityModal(eusilcS, eusilcP, "pl030", "ecoStat")
data(eusilcS)
data(eusilcP)
m1 <- meanWt(eusilcS$age, eusilcS$rb050)
m2 <- mean(eusilcP$age)
utilityIndicator(m1, m2)
```

weighted_estimators

Weighted mean, variance, covariance matrix and correlation matrix

Description

Compute mean, variance, covariance matrix and correlation matrix, taking into account sample weights.

- meanWt: 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. Implemented methods for this generic are:
  - meanWt.default(x,weights,na.rm=TRUE,...)
  - meanWt.dataObj(x,vars,na.rm=TRUE,...)
- varWt: calls var(x,na.rm=na.rm) if weights is missing. Implemented methods for this generic are:
  - varWt.default(x,weights,na.rm=TRUE,...)
  - varWt.dataObj(x,vars,na.rm=TRUE,...)
- covWt and covWt: always remove missing values pairwise and call cov and cor, respectively, if weights is missing. Implemented methods for these generics are:
  - covWt.default(x,y,weights,...)
  - covWt.matrix(x,weights,...)
  - covWt.data.frame(x,weights,...)
  - covWt.dataObj(x,vars,...)
  - corWt.default(x,y,weights,...)
  - corWt.matrix(x,weights,...)
  - corWt.data.frame(x,weights,...)
  - corWt.dataObj(x,vars,...)

The additional parameters are now described:

- y: a numeric vector. If missing, this defaults to x.
- vars: a character vector of variable names that should be used for the calculation.
- na.rm: a logical indicating whether any NA or NaN values should be removed from x before computation. Note that the default is TRUE.
- weights: an optional numeric vector containing sample weights.

Usage

```
meanWt(x, ...)

varWt(x, ...)

covWt(x, ...)

corWt(x, ...)
```

Arguments

x	for meanWt and varWt, a numeric vector or an object of class dataObj. For covWt and corWt, a numeric vector, matrix, data.frame or dataObj. In case of a dataObj, weights are automatically used from the S4-object itself.
...	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.

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.

Note

meanWt, varWt, covWt and corWt all make use of slot weights of the input object if the dataObj-method is used.

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))

# dataObj-methods
inp <- specifyInput(data=eusilcS, hhid="db030", hhsize="hsize", strata="db040", weight="db09
meanWt(inp, vars="netIncome")
sqrt(varWt(inp, vars="netIncome"))
corWt(inp, vars=c("age", "netIncome"))
covWt(inp, vars=c("age", "netIncome"))
```

`whipple` *Whipple index (original and modified)*

Description

The function calculates the original and modified Whipple index to evaluate age heaping.

Usage

```
whipple(x, method = "standard", weight = NULL)
```

Arguments

`x`	numeric vector holding the age of persons
`method`	"standard" or "modified" Whipple index.
`weight`	numeric vector holding the weights of each person

Details

The original Whipple's index is obtained by summing the number of persons in the age range between 23 and 62, and calculating the ratio of reported ages ending in 0 or 5 to one-fifth of the total sample. A linear decrease in the number of persons of each age within the age range is assumed. Therefore, low ages (0-22 years) and high ages (63 years and above) are excluded from analysis since this assumption is not plausible.

When the digits 0 and 5 are not reported in the data, the original Whipple index varies between 0 and 100, 100 if no preference for 0 or 5 is within the data. When only the digits 0 and 5 are reported in the data it reaches a to a maximum of 500.

For the modified Whipple index, age heaping is calculated for all ten digits (0-9). For each digit, the degree of preference or avoidance can be determined for certain ranges of ages, and the modified Whipple index then is given by the absolute sum of these (indices - 1). The index is scaled between 0 and 1, therefore it is 1 if all age values end with the same digit and 0 it is distributed perfectly equally.

Value

The original or modified Whipple index.

Author(s)

Matthias Templ, Alexander Kowarik

References

Henry S. Shryock and Jacob S. Siegel, Methods and Materials of Demography (New York: Academic Press, 1976)

See Also

sprague

Examples

```
#Equally distributed
age <- sample(1:100, 5000, replace=TRUE)
whipple(age)
whipple(age,method="modified")

# Only 5 and 10
age5 <- sample(seq(0,100,by=5), 5000, replace=TRUE)
whipple(age5)
whipple(age5,method="modified")

#Only 10
age10 <- sample(seq(0,100,by=10), 5000, replace=TRUE)
whipple(age10)
whipple(age10,method="modified")
```