# Package 'simFrame'

February 20, 2015

**Version** 0.5.3

**Title** Simulation framework

**Date** 2014-03-03

**Depends** R (>= 2.14.0), Rcpp (>= 0.8.6), lattice, parallel

**Imports** methods, stats4

**Suggests** laeken, mvtnorm, robCompositions, sampling

**LinkingTo** Rcpp

**SystemRequirements** GNU make

**Description** A general framework for statistical simulation.

**License** GPL (>= 2)

**LazyLoad** yes

**Author** Andreas Alfons [aut, cre]

**Maintainer** Andreas Alfons <alfons@ese.eur.nl>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2014-03-03 10:43:34

## R topics documented:

simFrame-package          *Simulation framework*

## Description

A general framework for statistical simulation.

## Details

| | |
|---|---|
| Package: | simFrame |
| Version: | 0.5.3 |
| Date: | 2014-03-03 |
| Depends: | R (>= 2.14.0), Rcpp (>= 0.8.6), lattice, parallel |
| Imports: | methods, stats4 |
| Suggests: | laeken, mvtnorm, robCompositions, sampling |
| LinkingTo: | Rcpp |
| SystemRequirements: | GNU make |
| License: | GPL (>= 2) |
| LazyLoad: | yes |

Index:

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

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

Further information is available in the following vignettes:

simFrame-eusilc    Applications of Statistical Simulation in the Case of EU-SILC: Using the R Package simFrame (source, p

simFrame-intro    An Object-Oriented Framework for Statistical Simulation: The R Package simFrame (source, pdf)

**UML class diagram**

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

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

### Author(s)

Andreas Alfons [aut, cre]

Maintainer: Andreas Alfons <alfons@ese.eur.nl>

### References

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

---

accessors                    *Accessor and mutator functions for objects*

---

### Description

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

### Usage

```
getAdd(x)

getAux(x)
setAux(x, aux)

getCall(x, ...)

getCollect(x)
setCollect(x, collect)

getColnames(x)
setColnames(x, colnames)

getContControl(x)
setContControl(x, contControl)

getControl(x)

getDataControl(x)

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

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

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

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

getEpsilon(x)
setEpsilon(x, epsilon)

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

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

getGrouping(x)
setGrouping(x, grouping)

getIndices(x)

getIntoContamination(x)
setIntoContamination(x, intoContamination)

getK(x)
setK(x, k)

getLegend(x)

getNAControl(x)
setNAControl(x, NAControl)

getNArate(x)
setNArate(x, NArate)

getNr(x)

getNrep(x)

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

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

getSAE(x)
setSAE(x, SAE)

getSampleControl(x)

getSeed(x)

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

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

getSplit(x)

getTarget(x)
setTarget(x, target)

getValues(x)
```

## Arguments

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

intoContamination

    a logical indicating whether missing values should also be inserted into contaminated observations (see "NAControl").

k    a single positive integer giving the number of samples to be set up (see "VirtualSampleControl").

NAControl  an object of class "NAControl" (see "SimControl").

NArate   a numeric vector or matrix giving missing value rates (see "VirtualNAControl").

prob    a numeric vector giving probability weights (see "SampleControl" and "TwoStageControl").

SAE    a logical indicating whether small area estimation will be used in the simulation experiment (see "SimControl").

size    a non-negative integer or a vector of non-negative integers (see "DataControl", "SampleControl" and "TwoStageControl").

stage    optional integer; for certain slots of "TwoStageControl", this allows to access or modify only the list component for the specified stage. Use 1 for the first stage and 2 for the second stage.

target   a character vector specifying target columns (see "VirtualContControl" and "VirtualNAControl").

...    only used to allow for the stage argument in accessor and mutator methods for "TwoStageControl". Otherwise no additional arguments are available.

## Value

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

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

## Methods for function getAdd

  signature(x = "SimResults")

## Methods for functions getAux and setAux

  signature(x = "ContControl")

  signature(x = "NAControl")

## Methods for function getCall

  signature(x = "SampleSetup")

  signature(x = "SimResults")

  signature(x = "Strata")

## Methods for functions getCollect and setCollect

  signature(x = "SampleControl")

## Methods for function getColnames

  signature(x = "DataControl")

  signature(x = "SimResults")

**Methods for function setColnames**

    signature(x = "DataControl")

**Methods for functions getContControl and setContControl**

    signature(x = "SimControl")

**Methods for function getControl**

    signature(x = "SampleSetup")

    signature(x = "SimResults")

**Methods for function getDataControl**

    signature(x = "SimResults")

**Methods for function getDesign**

    signature(x = "SampleControl")

    signature(x = "TwoStageControl")

    signature(x = "SimControl")

    signature(x = "SimResults")

    signature(x = "Strata")

**Methods for function setDesign**

    signature(x = "SampleControl")

    signature(x = "TwoStageControl")

    signature(x = "SimControl")

**Methods for functions getDistribution and setDistribution**

    signature(x = "DataControl")

    signature(x = "DCARContControl")

**Methods for functions getDots and setDots**

    signature(x = "DataControl")

    signature(x = "DARContControl")

    signature(x = "DCARContControl")

    signature(x = "SampleControl")

    signature(x = "TwoStageControl")

    signature(x = "SimControl")

**Methods for function getEpsilon**

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

**Methods for function setEpsilon**

> signature(x = "VirtualContControl")

**Methods for functions getFun and setFun**

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

**Methods for functions getGrouping and setGrouping**

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

**Methods for function getIndices**

> signature(x = "SampleSetup")

**Methods for functions getIntoContamination and setIntoContamination**

> signature(x = "NAControl")

**Methods for functions getK and setK**

> signature(x = "VirtualSampleControl")

**Methods for function getLegend**

> signature(x = "Strata")

**Methods for functions getNAControl and setNAControl**

> signature(x = "SimControl")

**Methods for function getNArate**

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

**Methods for function setNArate**

    `signature(x = "VirtualNAControl")`

**Methods for function getNr**

    `signature(x = "Strata")`

**Methods for function getNrep**

    `signature(x = "SimResults")`

**Methods for function getProb**

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

**Methods for function setProb**

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

**Methods for functions getSAE and setSAE**

    `signature(x = "SimControl")`

**Methods for function getSampleControl**

    `signature(x = "SimResults")`

**Methods for function getSeed**

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

**Methods for function getSize**

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

**Methods for function setSize**

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

**Methods for function getSplit**

signature(x = "Strata")

**Methods for functions getTarget and setTarget**

signature(x = "VirtualContControl")

signature(x = "VirtualNAControl")

**Methods for function getValues**

signature(x = "SimResults")

signature(x = "Strata")

### Author(s)

Andreas Alfons

### References

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

### Examples

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

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

---

aggregate-methods            *Method for aggregating simulation results*

---

### Description

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

### Usage

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

**Arguments**

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

**Value**

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

**Details**

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

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

**Methods**

x = `"SimResults"` aggregate simulation results.

**Author(s)**

Andreas Alfons

**References**

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

**See Also**

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

**Examples**

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

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

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

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

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

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


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

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

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

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

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

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

---

BasicVector-class          *Class "BasicVector"*

---

### Description

Virtual class used internally for convenience.

## Objects from the Class

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

## Extends

Class `"OptBasicVector"`, directly.

## Methods

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

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

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

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

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

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

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

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

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

## UML class diagram

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

## Author(s)

Andreas Alfons

## Examples

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

---

clusterRunSimulation     *Run a simulation experiment on a cluster*

---

### Description

Generic function for running a simulation experiment on a cluster.

### Usage

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

### Arguments

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

## Details

Statistical simulation is embarrassingly parallel, hence computational performance can be increased by parallel computing. Since version 0.5.0, parallel computing in simFrame is implemented using the package parallel, which is part of the R base distribution since version 2.14.0 and builds upon work done for the contributed packages multicore and snow. Note that all objects and packages required for the computations (including simFrame) need to be made available on every worker process unless the worker processes are created by forking (see makeCluster).

In order to prevent problems with random numbers and to ensure reproducibility, random number streams should be used. With parallel, random number streams can be created via the function clusterSetRNGStream().

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

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

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

## Value

An object of class "SimResults".

## Methods

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

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

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

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

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

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

**Author(s)**

Andreas Alfons

**References**

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

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

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

Tierney, L., Rossini, A. and Li, N. (2009) snow: A Parallel Computing Framework for the R System. *International Journal of Parallel Programming*, **37**(1), 78–90.

**See Also**

makeCluster, clusterSetRNGStream, runSimulation, "SimControl", "SimResults", simBwplot, simDensityplot, simXyplot

**Examples**

```
## Not run:
## these examples requires at least a dual core processor


## design-based simulation
data(eusilcP)  #load data

# start cluster
cl <- makeCluster(2, type = "PSOCK")

# load package and data on workers
clusterEvalQ(cl, {
    library(simFrame)
    data(eusilcP)
})

# set up random number stream
clusterSetRNGStream(cl, iseed = "12345")

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

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

```
# export objects to workers
clusterExport(cl, c("sc", "cc", "sim"))

# run simulation on cluster
results <- clusterRunSimulation(cl, eusilcP,
    sc, contControl = cc, fun = sim)

# stop cluster
stopCluster(cl)

# explore results
head(results)
aggregate(results)
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)



## model-based simulation

# start cluster
cl <- makeCluster(2, type = "PSOCK")

# load package on workers
clusterEvalQ(cl, library(simFrame))

# set up random number stream
clusterSetRNGStream(cl, iseed = "12345")

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

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

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

# export objects to workers
clusterExport(cl, c("rgnorm", "means", "dc", "cc", "sim"))
```

```
# run simulation on cluster
results <- clusterRunSimulation(cl, dc, nrep = 100,
    contControl = cc, design = "group", fun = sim)

# stop cluster
stopCluster(cl)

# explore results
head(results)
aggregate(results)
plot(results, true = means)

## End(Not run)
```

---

clusterSetup                 *Set up multiple samples on a cluster*

---

#### Description

Generic function for setting up multiple samples on a cluster.

#### Usage

```
clusterSetup(cl, x, control, ...)

## S4 method for signature 'ANY,data.frame,SampleControl'
clusterSetup(cl, x, control)
```

#### Arguments

| | |
|---|---|
| cl | a cluster as generated by [makeCluster](). |
| x | the data.frame to sample from. |
| control | a control object inheriting from the virtual class "VirtualSampleControl" or a character string specifying such a control class (the default being "SampleControl"). |
| ... | if control is a character string or missing, the slots of the control object may be supplied as additional arguments. See ["SampleControl"]() for details on the slots. |

#### Details

A fundamental design principle of the framework in the case of design-based simulation studies is that the sampling procedure is separated from the simulation procedure. Two main advantages arise from setting up all samples in advance.

First, the repeated sampling reduces overall computation time dramatically in certain situations, since computer-intensive tasks like stratification need to be performed only once. This is particularly relevant for large population data. In close-to-reality simulation studies carried out in research projects in survey statistics, often up to 10000 samples are drawn from a population of millions of

individuals with stratified sampling designs. For such large data sets, stratification takes a considerable amount of time and is a very memory-intensive task. If the samples are taken on-the-fly, i.e., in every simulation run one sample is drawn, the function to take the stratified sample would typically split the population into the different strata in each of the 10000 simulation runs. If all samples are drawn in advance, on the other hand, the population data need to be split only once and all 10000 samples can be taken from the respective strata together.

Second, the samples can be stored permanently, which simplifies the reproduction of simulation results and may help to maximize comparability of results obtained by different partners in a research project. In particular, this is useful for large population data, when complex sampling techniques may be very time-consuming. In research projects involving different partners, usually different groups investigate different kinds of estimators. If the two groups use not only the same population data, but also the same previously set up samples, their results are highly comparable.

The computational performance of setting up multiple samples can be increased by parallel computing. Since version 0.5.0, parallel computing in simFrame is implemented using the package parallel, which is part of the R base distribution since version 2.14.0 and builds upon work done for the contributed packages multicore and snow. Note that all objects and packages required for the computations (including simFrame) need to be made available on every worker process unless the worker processes are created by forking (see makeCluster).

In order to prevent problems with random numbers and to ensure reproducibility, random number streams should be used. With parallel, random number streams can be created via the function clusterSetRNGStream().

The control class "SampleControl" is highly flexible and allows stratified sampling as well as sampling of whole groups rather than individuals with a specified sampling method. Hence it is often sufficient to implement the desired sampling method for the simple non-stratified case to extend the existing framework. See "SampleControl" for some restrictions on the argument names of such a function, which should return a vector containing the indices of the sampled observations.

Nevertheless, for very complex sampling procedures, it is possible to define a control class "MySampleControl" extending "VirtualSampleControl", and the corresponding method clusterSetup(cl, x, control) with signature 'ANY, data.frame, MySampleControl'. In order to optimize computational performance, it is necessary to efficiently set up multiple samples. Thereby the slot k of "VirtualSampleControl" needs to be used to control the number of samples, and the resulting object must be of class "SampleSetup".

### Value

An object of class "SampleSetup".

### Methods

cl = "ANY", x = "data.frame", control = "character" set up multiple samples on a cluster using a control class specified by the character string control. The slots of the control object may be supplied as additional arguments.

cl = "ANY", x = "data.frame", control = "missing" set up multiple samples on a cluster using a control object of class "SampleControl". Its slots may be supplied as additional arguments.

cl = "ANY", x = "data.frame", control = "SampleControl" set up multiple samples on a cluster as defined by the control object control.

**Author(s)**

Andreas Alfons

**References**

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

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

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

Tierney, L., Rossini, A. and Li, N. (2009) snow: A Parallel Computing Framework for the R System. *International Journal of Parallel Programming*, **37**(1), 78–90.

**See Also**

makeCluster, clusterSetRNGStream, setup, draw, "SampleControl", "TwoStageControl", "VirtualSampleControl", "SampleSetup"

**Examples**

```
## Not run:
# these examples require at least a dual core processor

# load data
data(eusilcP)

# start cluster
cl <- makeCluster(2, type = "PSOCK")

# load package and data on workers
clusterEvalQ(cl, {
        library(simFrame)
        data(eusilcP)
    })

# set up random number stream
clusterSetRNGStream(cl, iseed = "12345")

# simple random sampling
srss <- clusterSetup(cl, eusilcP, size = 20, k = 4)
summary(srss)
draw(eusilcP[, c("id", "eqIncome")], srss, i = 1)

# group sampling
gss <- clusterSetup(cl, eusilcP, grouping = "hid", size = 10, k = 4)
summary(gss)
draw(eusilcP[, c("hid", "id", "eqIncome")], gss, i = 2)
```

```
# stratified simple random sampling
ssrss <- clusterSetup(cl, eusilcP, design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(ssrss)
draw(eusilcP[, c("id", "region", "eqIncome")], ssrss, i = 3)

# stratified group sampling
sgss <- clusterSetup(cl, eusilcP, design = "region",
    grouping = "hid", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(sgss)
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgss, i = 4)

# stop cluster
stopCluster(cl)

## End(Not run)
```

---

contaminate                    *Contaminate data*

---

#### Description

Generic function for contaminating data.

#### Usage

```
contaminate(x, control, ...)

## S4 method for signature 'data.frame,ContControl'
contaminate(x, control, i)
```

#### Arguments

| | |
|---|---|
| x | the data to be contaminated. |
| control | a control object of a class inheriting from the virtual class "VirtualContControl" or a character string specifying such a control class (the default being "DCARContControl"). |
| i | an integer giving the element of the slot epsilon of control to be used as contamination level. |
| ... | if control is a character string or missing, the slots of the control object may be supplied as additional arguments. See "DCARContControl" and "DARContControl" for details on the slots. |

#### Details

With the control classes implemented in **simFrame**, contamination is modeled as a two-step process. The first step is to select observations to be contaminated, the second is to model the distribution of the outliers.

In order to extend the framework by a user-defined control class "MyContControl" (which must extend "VirtualContControl"), a method contaminate(x, control, i) with signature 'data.frame, MyContControl' needs to be implemented. In case the contaminated observations need to be identified at a later stage of the simulation, e.g., if conflicts with inserting missing values should be avoided, a logical indicator variable ".contaminated" should be added to the returned data set.

### Value

A data.frame containing the contaminated data. In addition, the column ".contaminated", which consists of logicals indicating the contaminated observations, is added to the data.frame.

### Methods

x = "data.frame", control = "character" contaminate data using a control class specified by the character string control. The slots of the control object may be supplied as additional arguments.

x = "data.frame", control = "ContControl" contaminate data as defined by the control object control.

x = "data.frame", control = "missing" contaminate data using a control object of class "ContControl". Its slots may be supplied as additional arguments.

### Note

Since version 0.3, contaminate no longer checks if the auxiliary variable with probability weights are numeric and contain only finite positive values (sample still throws an error in these cases). This has been removed to improve computational performance in simulation studies.

### Author(s)

Andreas Alfons

### References

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

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

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

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

### See Also

"DCARContControl", "DARContControl", "ContControl", "VirtualContControl"

## Examples

```
## distributed completely at random
data(eusilcP)
sam <- draw(eusilcP[, c("id", "eqIncome")], size = 20)

# using a control object
dcarc <- ContControl(target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000), type = "DCAR")
contaminate(sam, dcarc)

# supply slots of control object as arguments
contaminate(sam, target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000))


## distributed at random
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
foo <- generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))

# using a control object
darc <- DARContControl(target = "V2",
    epsilon = 0.2, fun = function(x) x * 100)
contaminate(foo, darc)

# supply slots of control object as arguments
contaminate(foo, "DARContControl", target = "V2",
    epsilon = 0.2, fun = function(x) x * 100)
```

---

ContControl                    *Create contamination control objects*

---

## Description

Create objects of a class inheriting from "ContControl".

## Usage

```
ContControl(..., type = c("DCAR", "DAR"))
```

## Arguments

| | |
|---|---|
| ... | arguments passed to new("DCARContControl", ...) or new("DARContControl", ...), as determined by type. |
| type | a character string specifying whether a control object of class "DCARContControl" or "DARContControl" should be created. |

## Value

If `type = "DCAR"`, an object of class `"DCARContControl"`.

If `type = "DAR"`, an object of class `"DARContControl"`.

## Note

This constructor exists mainly for back compatibility with early draft versions of `simFrame`.

## Author(s)

Andreas Alfons

## See Also

`"DCARContControl"`, `"DARContControl"`, `"ContControl"`

## Examples

```
## distributed completely at random
data(eusilcP)
sam <- draw(eusilcP[, c("id", "eqIncome")], size = 20)
dcarc <- ContControl(target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000), type = "DCAR")
contaminate(sam, dcarc)

## distributed at random
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
foo <- generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
darc <- ContControl(target = "V2", epsilon = 0.2,
    fun = function(x) x * 100, type = "DAR")
contaminate(foo, darc)
```

---

ContControl-class          *Class "ContControl"*

---

## Description

Virtual class for controlling contamination in a simulation experiment (used internally).

## Objects from the Class

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

**Slots**

target: Object of class "OptCharacter"; a character vector specifying specifying the variables (columns) to be contaminated, or NULL to contaminate all variables (except the additional ones generated internally).

epsilon: Object of class "numeric" giving the contamination levels.

grouping: Object of class "character" specifying a grouping variable (column) to be used for contaminating whole groups rather than individual observations.

aux: Object of class "character" specifying an auxiliary variable (column) whose values are used as probability weights for selecting the items (observations or groups) to be contaminated.

**Extends**

Class "VirtualContControl", directly. Class "OptContControl", by class "VirtualContControl", distance 2.

**Accessor and mutator methods**

In addition to the accessor and mutator methods for the slots inherited from "VirtualContControl", the following are available:

getGrouping signature(x = "ContControl"): get slot grouping.

setGrouping signature(x = "ContControl"): set slot grouping.

getAux signature(x = "ContControl"): get slot aux.

setAux signature(x = "ContControl"): set slot aux.

**Methods**

In addition to the methods inherited from "VirtualContControl", the following are available:

contaminate signature(x = "data.frame",       control = "ContControl"): contaminate data.

show signature(object = "ContControl"): print the object on the R console.

**UML class diagram**

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

**Note**

The slot grouping was named group prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named getGroup already exists.

**Author(s)**

Andreas Alfons

**References**

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

**See Also**

"DCARContControl", "DARContControl", "VirtualContControl", contaminate

**Examples**

```
showClass("ContControl")
```

---

DARContControl-class    *Class "DARContControl"*

---

**Description**

Class for controlling contamination in a simulation experiment. The values of the contaminated observations will be distributed at random (*DAR*), i.e., they will depend on on the original values.

**Objects from the Class**

Objects can be created by calls of the form new("DARContControl", ...), DARContControl(...) or ContControl(..., type="DAR").

**Slots**

target: Object of class "OptCharacter"; a character vector specifying specifying the variables (columns) to be contaminated, or NULL to contaminate all variables (except the additional ones generated internally).

epsilon: Object of class "numeric" giving the contamination levels.

grouping: Object of class "character" specifying a grouping variable (column) to be used for contaminating whole groups rather than individual observations.

aux: Object of class "character" specifying an auxiliary variable (column) whose values are used as probability weights for selecting the items (observations or groups) to be contaminated.

fun: Object of class "function" generating the values of the contamination data. The original values of the observations to be contaminated will be passed as its first argument. Furthermore, it should return an object that can be coerced to a data.frame, containing the contamination data.

dots: Object of class "list" containing additional arguments to be passed to fun.

**Extends**

Class "ContControl", directly. Class "VirtualContControl", by class "ContControl", distance 2. Class "OptContControl", by class "ContControl", distance 3.

## Details

With this control class, contamination is modeled as a two-step process. The first step is to select observations to be contaminated, the second is to model the distribution of the outliers. In this case, the original values will be modified by the function given by slot fun, i.e., values of the contaminated observations will depend on on the original values.

## Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from `"ContControl"`, the following are available:

getFun signature(x = "DARContControl"): get slot fun.

setFun signature(x = "DARContControl"): set slot fun.

getDots signature(x = "DARContControl"): get slot dots.

setDots signature(x = "DARContControl"): set slot dots.

## Methods

Methods are inherited from `"ContControl"`.

## UML class diagram

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

## Note

The slot grouping was named group prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named getGroup already exists.

## Author(s)

Andreas Alfons

## References

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

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

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

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

## See Also

*"DCARContControl"*, *"ContControl"*, *"VirtualContControl"*, contaminate

## Examples

```
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
foo <- generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
cc <- DARContControl(target = "V2",
    epsilon = 0.2, fun = function(x) x * 100)
contaminate(foo, cc)
```

---

DataControl-class          *Class "DataControl"*

---

## Description

Class for controlling model-based generation of data.

## Objects from the Class

Objects can be created by calls of the form new("DataControl", ...) or DataControl(...).

## Slots

size: Object of class "numeric" giving the number of observations to be generated.

distribution: Object of class "function" generating the data, e.g., rnorm (the default) or rmvnorm. It should take a positive integer as its first argument, giving the number of observations to be generated, and return an object that can be coerced to a data.frame.

dots: Object of class "list" containing additional arguments to be passed to distribution.

colnames: Object of class "OptCharacter" ; a character vector to be used as column names for the generated data.frame, or NULL.

## Extends

Class *"VirtualDataControl"*, directly. Class *"OptDataControl"*, by class "VirtualDataControl", distance 2.

## Accessor and mutator methods

getSize signature(x = "DataControl"): get slot size.

setSize signature(x = "DataControl"): set slot size.

getDistribution signature(x = "DataControl"): get slot distribution.

setDistribution signature(x = "DataControl"): set slot distribution.

getDots signature(x = "DataControl"): get slot dots.

setDots signature(x = "DataControl"): set slot dots.

getColnames signature(x = "DataControl"): get slot colnames.

setColnames signature(x = "DataControl"): set slot colnames.

## Methods

In addition to the methods inherited from "VirtualDataControl", the following are available:

generate signature(control = "DataControl"): generate data.

show signature(object = "DataControl"): print the object on the R console.

## UML class diagram

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

## Author(s)

Andreas Alfons

## References

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

## See Also

"VirtualDataControl", generate

## Examples

```
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
dc <- DataControl(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
generate(dc)
```

DCARContControl-class    *Class "DCARContControl"*

### Description

Class for controlling contamination in a simulation experiment. The values of the contaminated observations will be distributed completely at random (*DCAR*), i.e., they will not depend on on the original values.

### Objects from the Class

Objects can be created by calls of the form new("DCARContControl", ...), DCARContControl(...) or ContControl(..., type="DCAR") (the latter exists mainly for back compatibility with early draft versions of simFrame).

### Slots

target: Object of class "OptCharacter"; a character vector specifying specifying the variables (columns) to be contaminated, or NULL to contaminate all variables (except the additional ones generated internally).

epsilon: Object of class "numeric" giving the contamination levels.

grouping: Object of class "character" specifying a grouping variable (column) to be used for contaminating whole groups rather than individual observations (the same values are used for all observations in the same group).

aux: Object of class "character" specifying an auxiliary variable (column) whose values are used as probability weights for selecting the items (observations or groups) to be contaminated.

distribution: Object of class "function" generating the values of the contamination data, e.g., [rnorm] (the default) or [rmvnorm]. It should take a non-negative integer as its first argument, giving the number of items to be created, and return an object that can be coerced to a data.frame, containing the contamination data.

dots: Object of class "list" containing additional arguments to be passed to distribution.

### Extends

Class "[ContControl]", directly. Class "[VirtualContControl]", by class "ContControl", distance 2. Class "[OptContControl]", by class "ContControl", distance 3.

### Details

With this control class, contamination is modeled as a two-step process. The first step is to select observations to be contaminated, the second is to model the distribution of the outliers. In this case, the values of the contaminated observations will be generated by the function given by slot fun and will not depend on on the original values.

**Accessor and mutator methods**

In addition to the accessor and mutator methods for the slots inherited from "`ContControl`", the following are available:

`getDistribution` signature(x = "DCARContControl"): get slot `distribution`.

`setDistribution` signature(x = "DCARContControl"): set slot `distribution`.

`getDots` signature(x = "DCARContControl"): get slot `dots`.

`setDots` signature(x = "DCARContControl"): set slot `dots`.

**Methods**

Methods are inherited from "`ContControl`".

**UML class diagram**

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

**Note**

The slot `grouping` was named `group` prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named `getGroup` already exists.

**Author(s)**

Andreas Alfons

**References**

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

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

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

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

**See Also**

"`DARContControl`", "`ContControl`", "`VirtualContControl`", `contaminate`

## Examples

```
data(eusilcP)
sam <- draw(eusilcP[, c("id", "eqIncome")], size = 20)
cc <- DCARContControl(target = "eqIncome", epsilon = 0.05,
    dots = list(mean = 5e+05, sd = 10000))
contaminate(sam, cc)
```

---

draw                          *Draw a sample*

---

### Description

Generic function for drawing a sample.

### Usage

```
draw(x, setup, ...)

## S4 method for signature 'data.frame,SampleSetup'
draw(x, setup, i = 1)

## S4 method for signature 'data.frame,VirtualSampleControl'
draw(x, setup)
```

### Arguments

| | |
|---|---|
| x | the data to sample from. |
| setup | an object of class "SampleSetup" containing previously set up samples, a control object inheriting from the virtual class "VirtualSampleControl" or a character string specifying such a control class (the default being "SampleControl"). |
| i | an integer specifying which one of the previously set up samples should be drawn. |
| ... | if setup is a character string or missing, the slots of the control object may be supplied as additional arguments. See "SampleControl" for details on the slots. |

### Value

A data.frame containing the sampled observations. In addition, the column ".weight", which consists of the sample weights, is added to the data.frame.

### Methods

x = "data.frame", setup = "character" draw a sample using a control class specified by the character string setup. The slots of the control object may be supplied as additional arguments.

x = "data.frame", setup = "missing" draw a sample using a control object of class "SampleControl".
  Its slots may be supplied as additional arguments.

x = "data.frame", setup = "SampleSetup" draw a previously set up sample.

x = "data.frame", setup = "VirtualSampleControl" draw a sample using a control object
  inheriting from the virtual class "VirtualSampleControl".

## Author(s)

Andreas Alfons

## References

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

## See Also

setup, "SampleSetup", "SampleControl", "TwoStageControl", "VirtualSampleControl"

## Examples

```
## load data
data(eusilcP)

## simple random sampling
draw(eusilcP[, c("id", "eqIncome")], size = 20)

## group sampling
draw(eusilcP[, c("hid", "id", "eqIncome")],
    grouping = "hid", size = 10)

## stratified simple random sampling
draw(eusilcP[, c("id", "region", "eqIncome")],
    design = "region", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))

## stratified group sampling
draw(eusilcP[, c("hid", "id", "region", "eqIncome")],
    design = "region", grouping = "hid",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))
```

---

eusilcP                    *Synthetic EU-SILC data*

---

## Description

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

**Usage**

```
data(eusilcP)
```

**Format**

A `data.frame` with 58 654 observations on the following 28 variables:

`hid` integer; the household ID.

`region` factor; the federal state in which the household is located (levels `Burgenland`, `Carinthia`, `Lower Austria`, `Salzburg`, `Styria`, `Tyrol`, `Upper Austria`, `Vienna` and `Vorarlberg`).

`hsize` integer; the number of persons in the household.

`eqsize` numeric; the equivalized household size according to the modified OECD scale.

`eqIncome` numeric; a simplified version of the equivalized household income.

`pid` integer; the personal ID.

**id** the household ID combined with the personal ID. The first five digits represent the household ID, the last two digits the personal ID (both with leading zeros).

`age` integer; the person's age.

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

`ecoStat` factor; the person's economic status (levels 1 = working full time, 2 = working part time, 3 = unemployed, 4 = pupil, student, further training or unpaid work experience or in compulsory military or community service, 5 = in retirement or early retirement or has given up business, 6 = permanently disabled or/and unfit to work or other inactive person, 7 = fulfilling domestic tasks and care responsibilities).

`citizenship` factor; the person's citizenship (levels `AT`, `EU` and `Other`).

`py010n` numeric; employee cash or near cash income (net).

`py050n` numeric; cash benefits or losses from self-employment (net).

`py090n` numeric; unemployment benefits (net).

`py100n` numeric; old-age benefits (net).

`py110n` numeric; survivor's benefits (net).

`py120n` numeric; sickness benefits (net).

`py130n` numeric; disability benefits (net).

`py140n` numeric; education-related allowances (net).

`hy040n` numeric; income from rental of a property or land (net).

`hy050n` numeric; family/children related allowances (net).

`hy070n` numeric; housing allowances (net).

`hy080n` numeric; regular inter-household cash transfer received (net).

`hy090n` numeric; interest, dividends, profit from capital investments in unincorporated business (net).

`hy110n` numeric; income received by people aged under 16 (net).

`hy130n` numeric; regular inter-household cash transfer paid (net).

`hy145n` numeric; repayments/receipts for tax adjustment (net).

`main` logical; indicates the main income holder (i.e., the person with the highest income) of each household.

## Details

The data set is used as population data in some of the examples in package simFrame. Note that it is included for illustrative purposes only. It consists of 25 000 households, hence it does not represent the true population sizes of Austria and its regions.

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

## Source

This is a synthetic data set based on Austrian EU-SILC data from 2006. The original sample was provided by Statistics Austria.

## References

Eurostat (2004) Description of target variables: Cross-sectional and longitudinal. *EU-SILC 065/04*, Eurostat.

## Examples

```
data(eusilcP)
summary(eusilcP)

strata <- stratify(eusilcP, c("region", "gender"))
summary(strata)
```

---

generate                    *Generate data*

---

## Description

Generic function for generating data based on a (distribution) model.

## Usage

```
generate(control, ...)

## S4 method for signature 'DataControl'
generate(control)
```

**Arguments**

control            a control object inheriting from the virtual class "VirtualDataControl" or a
                   character string specifying such a control class (the default being "DataControl").

...                if control is a character string or missing, the slots of the control object may be
                   supplied as additional arguments. See "DataControl" for details on the slots.

**Details**

The control class "DataControl" is quite simple but general. For user-defined data generation, it
often suffices to implement a function and use it as the distribution slot in the "DataControl"
object. See "DataControl" for some requirements for such a function.

However, if more specialized data generation models are required, the framework can be extended
by defining a control class "MyDataControl" extending "VirtualDataControl" and the corre-
sponding method generate(control) with signature 'MyDataControl'. If, e.g., a specific distri-
bution or mixture of distributions is frequently used in simulation experiments, a distinct control
class may be more convenient for the user.

**Value**

A data.frame.

**Methods**

control = "character" generate data using a control class specified by the character string
          control. The slots of the control object may be supplied as additional arguments.

control = "missing" generate data using a control object of class "DataControl". Its slots may
          be supplied as additional arguments.

control = "DataControl" generate data as defined by the control object control.

**Author(s)**

Andreas Alfons

**References**

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

**See Also**

"DataControl", "VirtualDataControl"

## Examples

```
require(mvtnorm)
mean <- rep(0, 2)
sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)

# using a control object
dc <- DataControl(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
generate(dc)

# supply slots of control object as arguments
generate(size = 10, distribution = rmvnorm,
    dots = list(mean = mean, sigma = sigma))
```

---

head-methods                 *Methods for returning the first parts of an object*

---

## Description

Return the first parts of an object.

## Usage

```
## S4 method for signature 'SampleSetup'
head(x, k = 6, n = 6, ...)

## S4 method for signature 'SimControl'
head(x)

## S4 method for signature 'SimResults'
head(x, ...)

## S4 method for signature 'Strata'
head(x, ...)

## S4 method for signature 'VirtualContControl'
head(x)

## S4 method for signature 'VirtualDataControl'
head(x)

## S4 method for signature 'VirtualNAControl'
head(x)

## S4 method for signature 'VirtualSampleControl'
head(x)
```

## Arguments

| | |
|---|---|
| x | an object. |
| k | for objects of class "SampleSetup", the number of set up samples to be kept in the resulting object. |
| n | for objects of class "SampleSetup", the number of indices to be kept in each of the set up samples in the resulting object. |
| ... | additional arguments to be passed down to methods. |

## Value

An object of the same class as x, but in general smaller. See the "Methods" section below for details.

## Methods

signature(x = "SampleSetup") returns the first parts of set up samples. The first n indices of each of the first k set up samples are kept.

signature(x = "SimControl") currently returns the object itself.

signature(x = "SimResults") returns the first parts of simulation results. The method of [head](#) for the data.frame in slot values is thereby called.

signature(x = "Strata") returns the first parts of strata information. The method of [head](#) for the vector in slot values is thereby called and the slots split and size are adapted accordingly.

signature(x = "VirtualContControl") currently returns the object itself.

signature(x = "VirtualDataControl") currently returns the object itself.

signature(x = "VirtualNAControl") currently returns the object itself.

signature(x = "VirtualSampleControl") currently returns the object itself.

## Author(s)

Andreas Alfons

## References

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

## See Also

[head](#), ["SampleSetup"](#), ["SimResults"](#), ["Strata"](#)

## Examples

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)
# get the first 10 indices of each of the first 5 samples
head(set, k = 5, n = 10)

## class "Strata"
# set up samples using group sampling
strata <- stratify(eusilcP, "region")
summary(strata)
# get strata information for the first 10 observations
head(strata, 10)
```

---

inclusionProb                     *Inclusion probabilities*

---

## Description

Get the first-order inclusion probabilities from a vector of probability weights.

## Usage

```
inclusionProb(prob, size)
```

## Arguments

| | |
|---|---|
| prob | a numeric vector of non-negative probability weights. |
| size | a non-negative integer giving the sample size. |

## Value

A numeric vector of the first-order inclusion probabilities.

## Note

This is a faster C++ implementation of inclusionprobabilities from package sampling.

## Author(s)

Andreas Alfons

## See Also

setup, "SampleSetup"

## Examples

```
pweights <- sample(1:5, 25, replace = TRUE)
inclusionProb(pweights, 10)
```

---

length-methods            *Methods for getting the length of an object*

---

### Description

Get the length of an object.

### Usage

```
## S4 method for signature 'SampleSetup'
length(x)

## S4 method for signature 'VirtualContControl'
length(x)

## S4 method for signature 'VirtualNAControl'
length(x)

## S4 method for signature 'VirtualSampleControl'
length(x)
```

### Arguments

x                 an object.

### Value

An integer giving the length of the object. See the "Methods" section below for details.

### Methods

signature(x = "SampleSetup") get the number of set up samples.

signature(x = "VirtualContControl") get the number of contamination levels to be used.

signature(x = "VirtualNAControl") get the number of missing value rates to be used (the length in case of a vector in slot NArate or the number of rows in case of a matrix).

signature(x = "VirtualSampleControl") get the number of samples to be set up.

### Author(s)

Andreas Alfons

**See Also**

[length](#)

**Examples**

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)
length(set)

## class "ContControl"
cc <- ContControl(target = "eqIncome",
    epsilon = c(0, 0.0025, 0.005, 0.0075, 0.01),
    dots = list(mean = 5e+05, sd = 10000))
length(cc)

## class "NAControl"
nc <- NAControl(target = "eqIncome", NArate = c(0.1, 0.2, 0.3))
length(nc)
```

---

NAControl-class          *Class "NAControl"*

---

**Description**

Class for controlling the insertion of missing values in a simulation experiment.

**Objects from the Class**

Objects can be created by calls of the form new("NAControl", ...) or NAControl(...).

**Slots**

target: Object of class "OptCharacter"; a character vector specifying the variables (columns) in which missing values should be inserted, or NULL to insert missing values in all variables (except the additional ones generated internally).

NArate: Object of class "NumericMatrix" giving the missing value rates, which may be selected individually for the target variables. In case of a vector, the same missing value rates are used for all target variables. In case of a matrix, on the other hand, the missing value rates to be used for each target variable are given by the respective column.

grouping: Object of class "character" specifying a grouping variable (column) to be used for setting whole groups to NA rather than individual values.

**aux:** Object of class "character" specifying auxiliary variables (columns) whose values are used as probability weights for selecting the values to be set to NA in the respective target variables. If only one variable (column) is specified, it is used for all target variables.

**intoContamination:** Object of class "logical" indicating whether missing values should also be inserted into contaminated observations. The default is to insert missing values only into non-contaminated observations.

### Extends

Class "VirtualNAControl", directly. Class "OptNAControl", by class "VirtualNAControl", distance 2.

### Accessor and mutator methods

In addition to the accessor and mutator methods for the slots inherited from "VirtualNAControl", the following are available:

getGrouping signature(x = "NAControl"): get slot grouping.

setGrouping signature(x = "NAControl"): set slot grouping.

getAux signature(x = "NAControl"): get slot aux.

setAux signature(x = "NAControl"): set slot aux.

getIntoContamination signature(x = "NAControl"): get slot intoContamination.

setIntoContamination signature(x = "NAControl"): set slot intoContamination.

### Methods

In addition to the methods inherited from "VirtualNAControl", the following are available:

setNA signature(x = "data.frame",        control = "NAControl"): set missing values.

show signature(object = "NAControl"): print the object on the R console.

### UML class diagram

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

### Note

Since version 0.3, this control class now allows to specify an auxiliary variable with probability weights for each target variable.

The slot grouping was named group prior to version 0.2. Renaming the slot was necessary since accessor and mutator functions were introduced in this version and a function named getGroup already exists.

### Author(s)

Andreas Alfons

### References

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

### See Also

"VirtualNAControl", setNA

### Examples

```
data(eusilcP)
eusilcP$age[eusilcP$age < 0] <- 0  # this actually occurs
sam <- draw(eusilcP[, c("id", "age", "eqIncome")], size = 20)

## missing completely at random
mcarc <- NAControl(target = "eqIncome", NArate = 0.2)
setNA(sam, mcarc)

## missing at random
marc <- NAControl(target = "eqIncome", NArate = 0.2, aux = "age")
setNA(sam, marc)

## missing not at random
mnarc <- NAControl(target = "eqIncome",
    NArate = 0.2, aux = "eqIncome")
setNA(sam, mnarc)
```

---

NumericMatrix-class     *Class "NumericMatrix"*

---

### Description

Virtual class used internally for convenience.

### Objects from the Class

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

### Methods

No methods defined with class "NumericMatrix" in the signature.

### UML class diagram

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

## Author(s)

Andreas Alfons

## Examples

```
showClass("NumericMatrix")
```

---

OptBasicVector-class     *Class "OptBasicVector"*

---

## Description

Virtual class used internally for convenience.

## Objects from the Class

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

## Methods

No methods defined with class `"OptBasicVector"` in the signature.

## UML class diagram

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

## Author(s)

Andreas Alfons

## See Also

`"SampleControl"`

## Examples

```
showClass("OptBasicVector")
```

---

OptCall-class *Class "OptCall"*

---

### Description

Virtual class used internally for convenience.

### Objects from the Class

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

### Methods

No methods defined with class "OptCall" in the signature.

### Author(s)

Andreas Alfons

### Examples

```
showClass("OptCall")
```

---

OptCharacter-class *Class "OptCharacter"*

---

### Description

Virtual class used internally for convenience.

### Objects from the Class

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

### Methods

No methods defined with class "OptCharacter" in the signature.

### UML class diagram

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

### Author(s)

Andreas Alfons

## Examples

```
showClass("OptCharacter")
```

---

OptContControl-class   *Class "OptContControl"*

---

### Description

Virtual class used internally for convenience.

### Objects from the Class

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

### Methods

No methods defined with class `"OptContControl"` in the signature.

### UML class diagram

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

### Author(s)

Andreas Alfons

### See Also

`"SimControl"`

### Examples

```
showClass("OptContControl")
```

---

OptDataControl-class   *Class "OptDataControl"*

---

### Description

Virtual class used internally for convenience.

### Objects from the Class

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

### Methods

No methods defined with class `"OptDataControl"` in the signature.

### UML class diagram

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

### Author(s)

Andreas Alfons

### See Also

`"SimResults"`

### Examples

```
showClass("OptDataControl")
```

---

OptNAControl-class   *Class "OptNAControl"*

---

### Description

Virtual class used internally for convenience.

### Objects from the Class

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

### Methods

No methods defined with class `"OptNAControl"` in the signature.

**UML class diagram**

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

**Author(s)**

Andreas Alfons

**See Also**

"SimControl"

**Examples**

```
showClass("OptNAControl")
```

---

OptNumeric-class          *Class "OptNumeric"*

---

**Description**

Virtual class used internally for convenience.

**Objects from the Class**

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

**Methods**

No methods defined with class "OptNumeric" in the signature.

**UML class diagram**

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

**Author(s)**

Andreas Alfons

**Examples**

```
showClass("OptNumeric")
```

---

```
OptSampleControl-class
```
*Class "OptSampleControl"*

---

### Description

Virtual class used internally for convenience.

### Objects from the Class

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

### Methods

No methods defined with class `"OptSampleControl"` in the signature.

### UML class diagram

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

### Author(s)

Andreas Alfons

### See Also

`"SimResults"`

### Examples

```
showClass("OptSampleControl")
```

---

```
plot-methods
```
*Plot simulation results*

---

### Description

Plot simulation results. A suitable plot function is selected automatically, depending on the structure of the results.

### Usage

```
## S4 method for signature 'SimResults,missing'
plot(x, y , ...)
```

## Arguments

x                    the simulation results.

y                    not used.

...                 further arguments to be passed to the selected plot function.

## Value

An object of class `"trellis"`. The `update` method can be used to update components of the object and the `print` method (usually called by default) will plot it on an appropriate plotting device.

## Details

The results of simulation experiments with at most one contamination level and at most one missing value rate are visualized by (conditional) box-and-whisker plots. For simulations involving different contamination levels or missing value rates, the average results are plotted against the contamination levels or missing value rates.

## Methods

x = `"SimResults"`, y = `"missing"` plot simulation results.

## Author(s)

Andreas Alfons

## References

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

## See Also

simBwplot, simDensityplot, simXyplot, `"SimResults"`

## Examples

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

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

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

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

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)




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

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

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

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

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

## plot results
plot(results, true = means)
```

---

runSimulation                    *Run a simulation experiment*

---

### Description

Generic function for running a simulation experiment.

**Usage**

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

runSim(...)
```

**Arguments**

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

**Details**

For convenience, the slots of control may be supplied as arguments.

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

For small area estimation, the following points have to be kept in mind. The design for splitting the data must be supplied and SAE must be set to TRUE. However, the data are not actually split into the specified domains. Instead, the whole data set (sample) is passed to fun. Also contamination

and missing values are added to the whole data (sample). Last, but not least, the function must have a `domain` argument so that the current domain can be extracted from the whole data (sample).

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

`runSim` is a wrapper for `runSimulation`.

## Value

An object of class `"SimResults"`.

## Methods

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

`x = "data.frame", setup = "missing", nrep = "missing", control = "SimControl"` run a simulation experiment based on real data without repetitions (probably useless, but for completeness).

`x = "data.frame", setup = "missing", nrep = "numeric", control = "SimControl"` run a simulation experiment based on real data with repetitions.

`x = "data.frame", setup = "SampleSetup", nrep = "missing", control = "SimControl"` run a design-based simulation experiment with previously set up samples.

`x = "data.frame", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl"` run a design-based simulation experiment.

`x = "VirtualDataControl", setup = "missing", nrep = "missing", control = "SimControl"` run a model-based simulation experiment without repetitions (probably useless, but for completeness).

`x = "VirtualDataControl", setup = "missing", nrep = "numeric", control = "SimControl"` run a model-based simulation experiment with repetitions.

`x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "missing", control = "SimControl"` run a simulation experiment using a mixed simulation design without repetitions (probably useless, but for completeness).

`x = "VirtualDataControl", setup = "VirtualSampleControl", nrep = "numeric", control = "SimControl"` run a simulation experiment using a mixed simulation design with repetitions.

## Author(s)

Andreas Alfons

## References

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

## See Also

[`"SimControl"`](#), [`"SimResults"`](#), [`simBwplot`](#), [`simDensityplot`](#), [`simXyplot`](#)

**Examples**

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

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

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

## run simulation and explore results
results <- runSimulation(eusilcP,
    sc, contControl = cc, fun = sim)
head(results)
aggregate(results)
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)




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

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

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

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

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

```
aggregate(results)
plot(results, true = means)
```

SampleControl-class     *Class "SampleControl"*

### Description

Class for controlling the setup of samples.

### Objects from the Class

Objects can be created by calls of the form new("SampleControl", ...) or SampleControl(...).

### Slots

design: Object of class "BasicVector" specifying variables (columns) to be used for stratified sampling.

grouping: Object of class "BasicVector" specifying a grouping variable (column) to be used for sampling whole groups rather than individual observations.

collect: Object of class "logical"; if a grouping variable is specified and this is FALSE (which is the default value), groups are sampled directly. If grouping variable is specified and this is TRUE, individuals are sampled in a first step. In a second step, all individuals that belong to the same group as any of the sampled individuals are collected and added to the sample. If no grouping variable is specified, this is ignored.

fun: Object of class "function" to be used for sampling (defaults to srs). It should return a vector containing the indices of the sampled items (observations or groups).

size: Object of class "OptNumeric"; an optional non-negative integer giving the number of items (observations or groups) to sample. In case of stratified sampling, a vector of non-negative integers, each giving the number of items to sample from the corresponding stratum, may be supplied.

prob: Object of class "OptBasicVector"; an optional numeric vector giving the probability weights, or a character string or logical vector specifying a variable (column) that contains the probability weights.

dots: Object of class "list" containing additional arguments to be passed to fun.

k: Object of class "numeric"; a single positive integer giving the number of samples to be set up.

### Details

There are some restrictions on the argument names of the function supplied to fun. If it needs population data as input, the corresponding argument should be called x and should expect a data.frame. If the sampling method only needs the population size as input, the argument should be called N. Note that fun is not expected to have both x and N as arguments, and that the latter is much faster for stratified sampling or group sampling. Furthermore, if the function has arguments for sample size and probability weights, they should be called size and prob, respectively. Note that a function with prob as its only argument is perfectly valid (for probability proportional to size sampling). Further arguments of fun may be supplied as a list via the slot dots.

**Extends**

Class "`VirtualSampleControl`", directly. Class "`OptSampleControl`", by class "VirtualSample-Control", distance 2.

**Accessor and mutator methods**

In addition to the accessor and mutator methods for the slots inherited from "`VirtualSampleControl`", the following are available:

getDesign signature(x = "SampleControl"): get slot design.

setDesign signature(x = "SampleControl"): set slot design.

getGrouping signature(x = "SampleControl"): get slot grouping.

setGrouping signature(x = "SampleControl"): set slot grouping.

getCollect signature(x = "SampleControl"): get slot collect.

setCollect signature(x = "SampleControl"): set slot collect.

getFun signature(x = "SampleControl"): get slot fun.

setFun signature(x = "SampleControl"): set slot fun.

getSize signature(x = "SampleControl"): get slot size.

setSize signature(x = "SampleControl"): set slot size.

getProb signature(x = "SampleControl"): get slot prob.

setProb signature(x = "SampleControl"): set slot prob.

getDots signature(x = "SampleControl"): get slot dots.

setDots signature(x = "SampleControl"): set slot dots.

**Methods**

In addition to the methods inherited from "`VirtualSampleControl`", the following are available:

clusterSetup signature(cl = "ANY", x = "data.frame",    control = "SampleControl"): set up multiple samples on a cluster.

setup signature(x = "data.frame",    control = "SampleControl"): set up multiple samples.

show signature(object = "SampleControl"): print the object on the R console.

**UML class diagram**

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

**Note**

The slots grouping and fun were named group and method, respectively, prior to version 0.2. Renaming the slots was necessary since accessor and mutator functions were introduced in this version and functions named getGroup, getMethod and setMethod already exist.

## Author(s)

Andreas Alfons

## References

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

## See Also

"VirtualSampleControl", "TwoStageControl", "SampleSetup", setup, draw

## Examples

```
data(eusilcP)

## simple random sampling
srsc <- SampleControl(size = 20)
draw(eusilcP[, c("id", "eqIncome")], srsc)

## group sampling
gsc <- SampleControl(grouping = "hid", size = 10)
draw(eusilcP[, c("hid", "hid", "eqIncome")], gsc)

## stratified simple random sampling
ssrsc <- SampleControl(design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))
draw(eusilcP[, c("id", "region", "eqIncome")], ssrsc)

## stratified group sampling
sgsc <- SampleControl(design = "region", grouping = "hid",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2))
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgsc)
```

---

SampleSetup-class          *Class "SampleSetup"*

---

## Description

Class for set up samples.

## Objects from the Class

Objects can be created by calls of the form new("SampleSetup", ...) or SampleSetup(...).

However, objects are expected to be created by the function setup or clusterSetup, these constructor functions are not supposed to be called by the user.

**Slots**

indices: Object of class "list"; each list element contains the indices of the sampled observa-
     tions.

prob: Object of class "numeric" giving the inclusion probabilities.

control: Object of class "VirtualSampleControl"; the control object used to set up the samples.

seed: Object of class "list" containing the seeds of the random number generator before and
     after setting up the samples, respectively (for replication purposes).

call: Object of class "SimCall"; the function call used to set up the samples, or NULL.

**Accessor methods**

getIndices signature(x = "SampleSetup"): get slot indices.

getProb signature(x = "SampleSetup"): get slot prob.

getControl signature(x = "SampleSetup"): get slot control.

getSeed signature(x = "SampleSetup"): get slot seed.

getCall signature(x = "SampleSetup"): get slot call.

**Methods**

clusterRunSimulation signature(cl = "ANY",      x = "data.frame", setup = "SampleSetup", nrep = "missi
     run a simulation experiment on a cluster.

draw signature(x = "data.frame",       setup = "SampleSetup"): draw a sample.

head signature(x = "SampleSetup"): returns the first parts of set up samples.

length signature(x = "SampleSetup"): get the number of set up samples.

runSimulation signature(x = "data.frame",      setup = "SampleSetup", nrep = "missing", control = "Sim
     run a simulation experiment.

show signature(object = "SampleSetup"): print set up samples on the R console.

summary signature(object = "SampleSetup"): produce a summary of set up samples.

tail signature(x = "SampleSetup"): returns the last parts of set up samples.

**UML class diagram**

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

**Note**

There are no mutator methods available since the slots are not supposed to be changed by the user.

Furthermore, the slot seed was added in version 0.2, and the slot control was added in version
0.3. Since the control object used to set up the samples is now stored, the redundant slots design,
grouping, collect and fun were removed. This has been done as preparation for additional control
classes for sampling, which will be introduced in future versions.

## Author(s)

Andreas Alfons

## References

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

## See Also

"SampleControl", "TwoStageControl", "VirtualSampleControl", setup, draw

## Examples

```
showClass("SampleSetup")
```

---

sampling                          *Random sampling*

---

## Description

Functions for random sampling.

## Usage

```
srs(N, size, replace = FALSE)

ups(N, size, prob, replace = FALSE)

brewer(prob, eps = 1e-06)

midzuno(prob, eps = 1e-06)

tille(prob, eps = 1e-06)
```

## Arguments

| | |
|---|---|
| N | a non-negative integer giving the number of observations from which to sample. |
| size | a non-negative integer giving the number of observations to sample. |
| prob | for ups, a numeric vector giving the probability weights (see sample). For tille and midzuno, a vector of inclusion probabilities (see inclusionProb). |
| replace | a logical indicating whether sampling should be performed with or without replacement. |
| eps | a numeric control value giving the desired accuracy. |

## Details

srs and ups are wrappers for simple random sampling and unequal probability sampling, respectively. Both functions make use of `sample`.

brewer, midzuno and tille perform Brewer's, Midzuno's and Tillé's method, respectively, for unequal probability sampling without replacement and fixed sample size.

## Value

An integer vector giving the indices of the sampled observations.

## Note

brewer, midzuno and tille are faster C++ implementations of `UPbrewer`, `UPmidzuno` and `UPtille`, respectively, from package sampling.

## Author(s)

Andreas Alfons

## References

Brewer, K. (1975), A simple procedure for sampling $\pi$ pswor, Australian Journal of Statistics, **17**(3), 166-172.

Midzuno, H. (1952) On the sampling system with probability proportional to sum of size. *Annals of the Institute of Statistical Mathematics*, **3**(2), 99–107.

Tillé, Y. (1996) An elimination procedure of unequal probability sampling without replacement. *Biometrika*, **83**(1), 238–241.

Deville, J.-C. and Tillé, Y. (1998) Unequal probability sampling without replacement through a splitting method. *Biometrika*, **85**(1), 89–101.

## See Also

*"SampleControl"*, *"TwoStageControl"*, setup, inclusionProb, sample, UPbrewer, UPmidzuno, UPtille

## Examples

```
## simple random sampling
# without replacement
srs(10, 5)
# with replacement
srs(5, 10, replace = TRUE)

## unequal probability sampling
# without replacement
ups(10, 5, prob = 1:10)
# with replacement
ups(5, 10, prob = 1:5, replace = TRUE)
```

```
## Brewer, Midzuno and Tille sampling
# define inclusion probabilities
prob <- c(0.2,0.7,0.8,0.5,0.4,0.4)
# Brewer sampling
brewer(prob)
# Midzuno sampling
midzuno(prob)
# Tille sampling
tille(prob)
```

---

setNA                       *Set missing values*

---

### Description

Generic function for inserting missing values into data.

### Usage

```
setNA(x, control, ...)

## S4 method for signature 'data.frame,NAControl'
setNA(x, control, i)
```

### Arguments

| | |
|---|---|
| x | the data in which missing values should be inserted. |
| control | a control object inheriting from the virtual class "VirtualNAControl" or a character string specifying such a control class (the default being "NAControl"). |
| i | an integer giving the element or row of the slot NArate of control to be used as missing value rate(s). |
| ... | if control is a character string or missing, the slots of the control object may be supplied as additional arguments. See "[NAControl](#)" for details on the slots. |

### Details

In order to extend the framework by a user-defined control class "MyNAControl" (which must extend "[VirtualNAControl](#)"), a method setNA(x, control, i) with signature 'data.frame, MyNAControl' needs to be implemented.

### Value

A data.frame containing the data with missing values.

## Methods

x = "data.frame", control = "character" set missing values using a control class specified
    by the character string control. The slots of the control object may be supplied as additional
    arguments.

x = "data.frame", control = "missing" set missing values using a control object of class
    "NAControl". Its slots may be supplied as additional arguments.

x = "data.frame", control = "NAControl" set missing values as defined by the control ob-
    ject control.

## Note

Since version 0.3, setNA no longer checks if auxiliary variable(s) with probability weights are
numeric and contain only finite positive values (sample still throws an error in these cases). This
has been removed to improve computational performance in simulation studies.

## Author(s)

Andreas Alfons

## References

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

## See Also

"NAControl", "VirtualNAControl"

## Examples

```
data(eusilcP)
eusilcP$age[eusilcP$age < 0] <- 0  # this actually occurs
sam <- draw(eusilcP[, c("id", "age", "eqIncome")], size = 20)


## using control objects
# missing completely at random
mcarc <- NAControl(target = "eqIncome", NArate = 0.2)
setNA(sam, mcarc)

# missing at random
marc <- NAControl(target = "eqIncome", NArate = 0.2, aux = "age")
setNA(sam, marc)

# missing not at random
mnarc <- NAControl(target = "eqIncome",
    NArate = 0.2, aux = "eqIncome")
setNA(sam, mnarc)
```

```
## supply slots of control object as arguments
# missing completely at random
setNA(sam, target = "eqIncome", NArate = 0.2)

# missing at random
setNA(sam, target = "eqIncome", NArate = 0.2, aux = "age")

# missing not at random
setNA(sam, target = "eqIncome", NArate = 0.2, aux = "eqIncome")
```

---

setup                      *Set up multiple samples*

---

### Description

Generic function for setting up multiple samples.

### Usage

```
setup(x, control, ...)

## S4 method for signature 'data.frame,SampleControl'
setup(x, control)
```

### Arguments

| | |
|---|---|
| x | the data to sample from. |
| control | a control object inheriting from the virtual class "VirtualSampleControl" or a character string specifying such a control class (the default being "SampleControl"). |
| ... | if control is a character string or missing, the slots of the control object may be supplied as additional arguments. See "SampleControl" for details on the slots. |

### Details

A fundamental design principle of the framework in the case of design-based simulation studies is that the sampling procedure is separated from the simulation procedure. Two main advantages arise from setting up all samples in advance.

First, the repeated sampling reduces overall computation time dramatically in certain situations, since computer-intensive tasks like stratification need to be performed only once. This is particularly relevant for large population data. In close-to-reality simulation studies carried out in research projects in survey statistics, often up to 10000 samples are drawn from a population of millions of individuals with stratified sampling designs. For such large data sets, stratification takes a considerable amount of time and is a very memory-intensive task. If the samples are taken on-the-fly, i.e., in every simulation run one sample is drawn, the function to take the stratified sample would typically split the population into the different strata in each of the 10000 simulation runs. If all samples are

drawn in advance, on the other hand, the population data need to be split only once and all 10000 samples can be taken from the respective strata together.

Second, the samples can be stored permanently, which simplifies the reproduction of simulation results and may help to maximize comparability of results obtained by different partners in a research project. In particular, this is useful for large population data, when complex sampling techniques may be very time-consuming. In research projects involving different partners, usually different groups investigate different kinds of estimators. If the two groups use not only the same population data, but also the same previously set up samples, their results are highly comparable.

The control class "SampleControl" is highly flexible and allows stratified sampling as well as sampling of whole groups rather than individuals with a specified sampling method. Hence it is often sufficient to implement the desired sampling method for the simple non-stratified case to extend the existing framework. See "SampleControl" for some restrictions on the argument names of such a function, which should return a vector containing the indices of the sampled observations.

Nevertheless, for very complex sampling procedures, it is possible to define a control class "MySampleControl" extending "VirtualSampleControl", and the corresponding method setup(x, control) with signature 'data.frame, MySampleControl'. In order to optimize computational performance, it is necessary to efficiently set up multiple samples. Thereby the slot k of "VirtualSampleControl" needs to be used to control the number of samples, and the resulting object must be of class "SampleSetup".

## Value

An object of class "SampleSetup".

## Methods

x = "data.frame", control = "character" set up multiple samples using a control class specified by the character string control. The slots of the control object may be supplied as additional arguments.

x = "data.frame", control = "missing" set up multiple samples using a control object of class "SampleControl". Its slots may be supplied as additional arguments.

x = "data.frame", control = "SampleControl" set up multiple samples as defined by the control object control.

## Author(s)

Andreas Alfons

## References

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

## See Also

simSample, draw, "SampleControl", "TwoStageControl", "VirtualSampleControl", "SampleSetup"

## Examples

```
set.seed(12345)  # for reproducibility
data(eusilcP)    # load data

## simple random sampling
srss <- setup(eusilcP, size = 20, k = 4)
summary(srss)
draw(eusilcP[, c("id", "eqIncome")], srss, i = 1)

## group sampling
gss <- setup(eusilcP, grouping = "hid", size = 10, k = 4)
summary(gss)
draw(eusilcP[, c("hid", "id", "eqIncome")], gss, i = 2)

## stratified simple random sampling
ssrss <- setup(eusilcP, design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(ssrss)
draw(eusilcP[, c("id", "region", "eqIncome")], ssrss, i = 3)

## stratified group sampling
sgss <- setup(eusilcP, design = "region",
    grouping = "hid", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(sgss)
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgss, i = 4)
```

---

simApply                *Apply a function to subsets*

---

## Description

Generic functions for applying a function to subsets of a data set.

## Usage

```
simApply(x, design, fun, ...)

simSapply(x, design, fun, ..., simplify = TRUE)
```

## Arguments

| | |
|---|---|
| x | the data.frame to be subsetted. |
| design | a character, logical or numeric vector specifying the variables (columns) used for subsetting. |
| fun | a function to be applied to the subsets. |
| simplify | a logical indicating whether the results should be simplified to a vector or matrix (if possible). |
| ... | additional arguments to be passed to fun. |

## Value

For simApply a data.frame.

For simSapply, a list, vector or matrix (see [sapply](#)).

## Methods for function simApply

x = "data.frame", design = "BasicVector", fun = "function" apply a function to subsets given by the variables (columns) in design.

x = "data.frame", design = "Strata", fun = "function" apply a function to subsets given by design.

## Methods for function simSapply

x = "data.frame", design = "BasicVector", fun = "function" apply a function to subsets given by the variables (columns) in design.

x = "data.frame", design = "Strata", fun = "function" apply a function to subsets given by design.

## Author(s)

Andreas Alfons

## See Also

[sapply](#)

## Examples

```
data(eusilcP)
eusilcP <- eusilcP[, c("region", "gender", "eqIncome")]

## returns data.frame
simApply(eusilcP, c("region", "gender"),
    function(x) median(x$eqIncome))

## returns vector
simSapply(eusilcP, c("region", "gender"),
    function(x) median(x$eqIncome))
```

---

simBwplot                    *Box-and-whisker plots*

---

## Description

Generic function for producing box-and-whisker plots.

## Usage

```
simBwplot(x, ...)

## S4 method for signature 'SimResults'
simBwplot(x, true = NULL, epsilon, NArate, select, ...)
```

## Arguments

| | |
|---|---|
| x | the object to be plotted. For plotting simulation results, this must be an object of class "SimResults". |
| true | a numeric vector giving the true values. If supplied, reference lines are drawn in the corresponding panels. |
| epsilon | a numeric vector specifying contamination levels. If supplied, the values corresponding to these contamination levels are extracted from the simulation results and plotted. |
| NArate | a numeric vector specifying missing value rates. If supplied, the values corresponding to these missing value rates are extracted from the simulation results and plotted. |
| select | a character vector specifying the columns to be plotted. It must be a subset of the colnames slot of x, which is the default. |
| ... | additional arguments to be passed down to methods and eventually to bwplot. |

## Details

For simulation results with multiple contamination levels or missing value rates, conditional box-and-whisker plots are produced.

## Value

An object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

## Methods

x = "SimResults" produce box-and-whisker plots of simulation results.

## Note

Functionality for producing conditional box-and-whisker plots was added in version 0.2. Prior to that, the function gave an error message if simulation results with multiple contamination levels or missing value rates were supplied.

## Author(s)

Andreas Alfons

**References**

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

**See Also**

simDensityplot, simXyplot, bwplot, "SimResults"

**Examples**

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

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

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

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

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
simBwplot(results, true = tv)



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

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

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

## function for simulation runs
```

```
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.02),
        median = median(x$value))
}

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

## plot results
simBwplot(results, true = means)
```

---

`SimControl-class`         *Class "SimControl"*

---

#### Description

Class for controlling how simulation runs are performed.

#### Objects from the Class

Objects can be created by calls of the form new("SimControl", ...) or SimControl(...).

#### Slots

contControl: Object of class "OptContControl"; a control object for contamination, or NULL.

NAControl: Object of class "OptNAControl"; a control object for inserting missing values, or NULL.

design: Object of class "character" specifying variables (columns) to be used for splitting the data into domains. The simulations, including contamination and the insertion of missing values (unless SAE=TRUE), are then performed on every domain.

fun: Object of class "function" to be applied in each simulation run.

dots: Object of class "list" containing additional arguments to be passed to fun.

SAE: Object of class "logical" indicating whether small area estimation will be used in the simulation experiment.

#### Details

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

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

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

### Accessor and mutator methods

getContControl signature(x = "SimControl"): get slot ContControl.

setContControl signature(x = "SimControl"): set slot ContControl.

getNAControl signature(x = "SimControl"): get slot NAControl.

setNAControl signature(x = "SimControl"): set slot NAControl.

getDesign signature(x = "SimControl"): get slot design.

setDesign signature(x = "SimControl"): set slot design.

getFun signature(x = "SimControl"): get slot fun.

setFun signature(x = "SimControl"): set slot fun.

getDots signature(x = "SimControl"): get slot dots.

setDots signature(x = "SimControl"): set slot dots.

getSAE signature(x = "SimControl"): get slot SAE.

setSAE signature(x = "SimControl"): set slot SAE.

### Methods

clusterRunSimulation signature(cl = "ANY",     x = "data.frame", setup = "missing", nrep = "numeric",
    run a simulation experiment on a cluster.

clusterRunSimulation signature(cl = "ANY",     x = "data.frame", setup = "VirtualSampleControl", nrep
    run a simulation experiment on a cluster.

clusterRunSimulation signature(cl = "ANY",     x = "data.frame", setup = "SampleSetup", nrep = "missin
    run a simulation experiment on a cluster.

clusterRunSimulation signature(cl = "ANY",     x = "VirtualDataControl", setup = "missing", nrep = "nu
    run a simulation experiment on a cluster.

clusterRunSimulation signature(cl = "ANY",     x = "VirtualDataControl", setup = "VirtualSampleContro
    run a simulation experiment on a cluster.

head signature(x = "SimControl"): currently returns the object itself.

runSimulation signature(x = "data.frame",     setup = "VirtualSampleControl", nrep = "missing",     c
    run a simulation experiment.

runSimulation signature(x = "data.frame",     setup = "SampleSetup", nrep = "missing", control = "SimC
    run a simulation experiment.

runSimulation signature(x = "data.frame",     setup = "missing", nrep = "numeric", control = "SimContr
    run a simulation experiment.

runSimulation signature(x = "data.frame",     setup = "missing", nrep = "missing", control = "SimContr
    run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",     setup = "missing", nrep = "numeric", control = "
    run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",     setup = "missing", nrep = "missing", control = "
    run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",     setup = "VirtualSampleControl", nrep = "numeric
    run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",     setup = "VirtualSampleControl", nrep = "missing
    run a simulation experiment.

show signature(object = "SimControl"): print the object on the R console.

summary signature(object = "SimControl"): currently returns the object itself.

tail signature(x = "SimControl"): currently returns the object itself.

### UML class diagram

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

### Author(s)

Andreas Alfons

### References

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

### See Also

runSimulation, "SimResults"

### Examples

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

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

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.02))
```

```
}

## combine these to "SimControl" object and run simulation
ctrl <- SimControl(contControl = cc, fun = sim)
results <- runSimulation(eusilcP, sc, control = ctrl)

## explore results
head(results)
aggregate(results)
tv <- mean(eusilcP$eqIncome)  # true population mean
plot(results, true = tv)




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

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

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

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

## combine these to "SimControl" object and run simulation
ctrl <- SimControl(contControl = cc, design = "group", fun = sim)
results <- runSimulation(dc, nrep = 50, control = ctrl)

## explore results
head(results)
aggregate(results)
plot(results, true = means)
```

---

simDensityplot            *Kernel density plots*

---

**Description**

Generic function for producing kernel density plots.

**Usage**

```
simDensityplot(x, ...)

## S4 method for signature 'SimResults'
simDensityplot(x, true = NULL, epsilon, NArate, select, ...)
```

**Arguments**

| | |
|---|---|
| x | the object to be plotted. For plotting simulation results, this must be an object of class "SimResults". |
| true | a numeric vector giving the true values. If supplied, reference lines are drawn in the corresponding panels. |
| epsilon | a numeric vector specifying contamination levels. If supplied, the values corresponding to these contamination levels are extracted from the simulation results and plotted. |
| NArate | a numeric vector specifying missing value rates. If supplied, the values corresponding to these missing value rates are extracted from the simulation results and plotted. |
| select | a character vector specifying the columns to be plotted. It must be a subset of the colnames slot of x, which is the default. |
| ... | additional arguments to be passed down to methods and eventually to densityplot. |

**Details**

For simulation results with multiple contamination levels or missing value rates, conditional kernel density plots are produced.

**Value**

An object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

**Methods**

x = "SimResults" produce kernel density plots of simulation results.

**Note**

Functionality for producing conditional kernel density plots was added in version 0.2. Prior to that, the function gave an error message if simulation results with multiple contamination levels or missing value rates were supplied.

## Author(s)

Andreas Alfons

## References

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

## See Also

simBwplot, simXyplot, densityplot, "SimResults"

## Examples

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

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

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

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

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
simDensityplot(results, true = tv)



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

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

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
```

```
cc <- DCARContControl(target = "value",
    epsilon = 0.02, dots = list(mean = 15))

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

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

## plot results
simDensityplot(results, true = means)
```

---

SimResults-class             *Class "SimResults"*

---

### Description

Class for simulation results.

### Objects from the Class

Objects can be created by calls of the form new("SimResults", ...) or SimResults(...).

However, objects are expected to be created by the function [runSimulation](#) or [clusterRunSimulation](#), these constructor functions are not supposed to be called by the user.

### Slots

values: Object of class "data.frame" containing the simulation results.

add: Object of class "list" containing additional simulation results, e.g., statistical models.

design: Object of class "character" giving the variables (columns) defining the domains used in the simulation experiment.

colnames: Object of class "character" giving the names of the columns of values that contain the actual simulation results.

epsilon: Object of class "numeric" containing the contamination levels used in the simulation experiment.

NArate: Object of class "NumericMatrix" containing the missing value rates used in the simulation experiment.

dataControl: Object of class "OptDataControl"; the control object used for data generation in model-based simulation, or NULL.

sampleControl: Object of class "OptSampleControl"; the control object used for sampling in design-based simulation, or NULL.

nrep: Object of class "numeric" giving the number of repetitions of the simulation experiment (for model-based simulation or simulation based on real data).

control: Object of class "SimControl"; the control object used for running the simulations.

seed: Object of class "list" containing the seeds of the random number generator before and after the simulation experiment, respectively (for replication of the results).

call: Object of class "SimCall"; the function call used to run the simulation experiment, or NULL.

### Accessor methods

getValues signature(x = "SimResults"): get slot values.

getAdd signature(x = "SimResults"): get slot add.

getDesign signature(x = "SimResults"): get slot design.

getColnames signature(x = "SimResults"): get slot colnames.

getEpsilon signature(x = "SimResults"): get slot epsilon.

getNArate signature(x = "SimResults"): get slot NArate.

getDataControl signature(x = "SimResults"): get slot dataControl.

getSampleControl signature(x = "SimResults"): get slot sampleControl.

getNrep signature(x = "SimResults"): get slot nrep.

getControl signature(x = "SimResults"): get slot control.

getSeed signature(x = "SimResults"): get slot seed.

getCall signature(x = "SimResults"): get slot call.

### Methods

aggregate signature(x = "SimResults"): aggregate simulation results.

head signature(x = "SimResults"): returns the first parts of simulation results.

plot signature(x = "SimResults", y = "missing"): selects a suitable graphical representation of the simulation results automatically.

show signature(object = "SimResults"): print simulation results on the R console.

simBwplot signature(x = "SimResults"): conditional box-and-whisker plot of simulation results.

simDensityplot signature(x = "SimResults"): conditional kernel density plot of simulation results.

simXyplot signature(x = "SimResults"): conditional x-y plot of simulation results.

summary signature(x = "SimResults"): produce a summary of simulation results.

tail signature(x = "SimResults"): returns the last parts of simulation results.

### UML class diagram

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

## Note

There are no mutator methods available since the slots are not supposed to be changed by the user.

Furthermore, the slots `dataControl`, `sampleControl`, `nrep` and `control` were added in version 0.3.

## Author(s)

Andreas Alfons

## References

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

## See Also

[runSimulation](runSimulation), [simBwplot](simBwplot), [simDensityplot](simDensityplot), [simXyplot](simXyplot)

## Examples

```
showClass("SimResults")
```

---

simSample                      *Set up multiple samples*

---

## Description

A convenience wrapper for setting up multiple samples using [setup](setup) with control class [SampleControl](SampleControl).

## Usage

```
simSample(x, design = character(), grouping = character(),
        collect = FALSE, fun = srs, size = NULL,
        prob = NULL, ..., k = 1)
```

## Arguments

| | |
|---|---|
| x | the `data.frame` to sample from. |
| design | a character, logical or numeric vector specifying variables (columns) to be used for stratified sampling. |
| grouping | a character string, single integer or logical vector specifying a grouping variable (column) to be used for sampling whole groups rather than individual observations. |

| | |
|---|---|
| collect | logical; if a grouping variable is specified and this is FALSE (which is the default value), groups are sampled directly. If grouping variable is specified and this is TRUE, individuals are sampled in a first step. In a second step, all individuals that belong to the same group as any of the sampled individuals are collected and added to the sample. If no grouping variable is specified, this is ignored. |
| fun | a function to be used for sampling (defaults to [srs](#)). It should return a vector containing the indices of the sampled items (observations or groups). |
| size | an optional non-negative integer giving the number of items (observations or groups) to sample. For stratified sampling, a vector of non-negative integers, each giving the number of items to sample from the corresponding stratum. |
| prob | an optional numeric vector giving the probability weights, or a character string or logical vector specifying a variable (column) that contains the probability weights. |
| ... | additional arguments to be passed to fun. |
| k | a single positive integer giving the number of samples to be set up. |

## Details

There are some restrictions on the argument names of the function supplied to fun. If it needs population data as input, the corresponding argument should be called x and should expect a data.frame. If the sampling method only needs the population size as input, the argument should be called N. Note that fun is not expected to have both x and N as arguments, and that the latter is much faster for stratified sampling or group sampling. Furthermore, if the function has arguments for sample size and probability weights, they should be called size and prob, respectively. Note that a function with prob as its only argument is perfectly valid (for probability proportional to size sampling). Further arguments of fun may be passed directly via the ... argument.

## Value

An object of class "SampleSetup".

## Author(s)

Andreas Alfons

## See Also

[setup](#), "[SampleControl](#)", "[SampleSetup](#)"

## Examples

```
data(eusilcP)

## simple random sampling
srss <- simSample(eusilcP, size = 20, k = 4)
summary(srss)
draw(eusilcP[, c("id", "eqIncome")], srss, i = 1)

## group sampling
```

```
gss <- simSample(eusilcP, grouping = "hid", size = 10, k = 4)
summary(gss)
draw(eusilcP[, c("hid", "id", "eqIncome")], gss, i = 2)

## stratified simple random sampling
ssrss <- simSample(eusilcP, design = "region",
    size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(ssrss)
draw(eusilcP[, c("id", "region", "eqIncome")], ssrss, i = 3)

## stratified group sampling
sgss <- simSample(eusilcP, design = "region",
    grouping = "hid", size = c(2, 5, 5, 3, 4, 5, 3, 5, 2), k = 4)
summary(sgss)
draw(eusilcP[, c("hid", "id", "region", "eqIncome")], sgss, i = 4)
```

---

| simXyplot | *X-Y plots* |
|---|---|

---

### Description

Generic function for producing x-y plots. For simulation results, the average results are plotted against the corresponding contamination levels or missing value rates.

### Usage

```
simXyplot(x, ...)

## S4 method for signature 'SimResults'
simXyplot(x, true = NULL, epsilon, NArate,
          select, cond = c("Epsilon", "NArate"),
          average = c("mean", "median"), ...)
```

### Arguments

| | |
|---|---|
| x | the object to be plotted. For plotting simulation results, this must be an object of class "SimResults". |
| true | a numeric vector giving the true values. If supplied, reference lines are drawn in the corresponding panels. |
| epsilon | a numeric vector specifying contamination levels. If supplied, the values corresponding to these contamination levels are extracted from the simulation results and plotted. |
| NArate | a numeric vector specifying missing value rates. If supplied, the values corresponding to these missing value rates are extracted from the simulation results and plotted. |
| select | a character vector specifying the columns to be plotted. It must be a subset of the colnames slot of x, which is the default. |

| | |
|---|---|
| cond | a character string; for simulation results with multiple contamination levels and multiple missing value rates, this specifies the column of the simulation results to be used for producing conditional x-y plots. If "Epsilon", conditional plots are produced for the different contamination levels. If "NArate", conditional plots are produced for the different missing value rates. The default is to use whichever results in less plots. |
| average | a character string specifying how the averages should be computed. Possible values are "mean" for the mean (the default) or "median" for the median. |
| ... | additional arguments to be passed down to methods and eventually to xyplot. |

### Details

For simulation results with multiple contamination levels and multiple missing value rates, conditional x-y plots are produced, as specified by cond.

### Value

An object of class "trellis". The update method can be used to update components of the object and the print method (usually called by default) will plot it on an appropriate plotting device.

### Methods

x = "SimResults" produce x-y plots of simulation results.

### Note

Functionality for producing conditional x-y plots (including the argument cond) was added in version 0.2. Prior to that, the function gave an error message if simulation results with multiple contamination levels and multiple missing value rates were supplied.

The argument average that specifies how the averages are computed was added in version 0.1.2. Prior to that, the mean has always been used.

### Author(s)

Andreas Alfons

### References

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

### See Also

simBwplot, simDensityplot, xyplot, "SimResults"

## Examples

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

## control objects for sampling and contamination
sc <- SampleControl(size = 500, k = 50)
cc <- DARContControl(target = "eqIncome",
    epsilon = seq(0, 0.05, by = 0.01),
    fun = function(x) x * 25)

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$eqIncome), trimmed = mean(x$eqIncome, 0.05))
}

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

## plot results
tv <- mean(eusilcP$eqIncome)  # true population mean
simXyplot(results, true = tv)




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

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

## control objects for data generation and contamination
means <- c(0, 0.25)
dc <- DataControl(size = 500, distribution = rgnorm,
    dots = list(means = means))
cc <- DCARContControl(target = "value",
    epsilon = seq(0, 0.05, by = 0.01),
    dots = list(mean = 15))

## function for simulation runs
sim <- function(x) {
    c(mean = mean(x$value),
        trimmed = mean(x$value, trim = 0.05),
        median = median(x$value))
}

## run simulation
results <- runSimulation(dc, nrep = 50,
```

```
        contControl = cc, design = "group", fun = sim)

## plot results
simXyplot(results, true = means)
```

---

Strata-class                    *Class "Strata"*

---

### Description

Class containing strata information for a data set.

### Objects from the Class

Objects can be created by calls of the form new("Strata", ...) or Strata(...).

However, objects are expected to be created by the function [stratify](#), these constructor functions are not supposed to be called by the user.

### Slots

values: Object of class "integer" giving the stratum number for each observation.

split: Object of class "list"; each list element contains the indices of the observations belonging to the corresponding stratum.

design: Object of class "character" giving the variables (columns) defining the strata.

nr: Object of class "integer" giving the stratum numbers.

legend: Object of class "data.frame" describing the strata.

size: Object of class "numeric" giving the stratum sizes.

call: Object of class "OptCall"; the function call used to stratify the data, or NULL.

### Accessor methods

getValues signature(x = "Strata"): get slot values.

getSplit signature(x = "Strata"): get slot split.

getDesign signature(x = "Strata"): get slot design.

getNr signature(x = "Strata"): get slot nr.

getLegend signature(x = "Strata"): get slot legend.

getSize signature(x = "Strata"): get slot size.

getCall signature(x = "Strata"): get slot call.

## Methods

head signature(x = "Strata"): returns the first parts of strata information.

show signature(object = "Strata"): print strata information on the R console.

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

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

summary signature(object = "Strata"): produce a summary of strata information.

tail signature(x = "Strata"): returns the last parts of strata information.

## UML class diagram

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

## Note

There are no mutator methods available since the slots are not supposed to be changed by the user.

## Author(s)

Andreas Alfons

## See Also

[stratify](#)

## Examples

```
showClass("Strata")
```

---

| stratify | *Stratify data* |
|---|---|

---

## Description

Generic function for stratifying data.

## Usage

```
stratify(x, design)
```

## Arguments

| | |
|---|---|
| x | the data.frame to be stratified. |
| design | a character, logical or numeric vector specifying the variables (columns) to be used for stratification. |

## Value

An object of class "Strata".

## Methods

x = "data.frame", design = "BasicVector" stratify data according to the variables (columns) given by design.

## Author(s)

Andreas Alfons

## See Also

"Strata"

## Examples

```
data(eusilcP)
strata <- stratify(eusilcP, c("region", "gender"))
summary(strata)
```

---

stratify-utilities          *Utility functions for stratifying data*

---

## Description

Generic utility functions for stratifying data. These are useful if not all the information of class "Strata" is necessary.

## Usage

```
getStrataLegend(x, design)

getStrataSplit(x, design, USE.NAMES = TRUE)

getStrataTable(x, design)

getStratumSizes(x, design, USE.NAMES = TRUE)

getStratumValues(x, design, split)
```

**Arguments**

| | |
|---|---|
| x | the `data.frame` to be stratified. For `getStratumSizes`, it is also possible to supply a list in which each list element contains the indices of the observations belonging to the corresponding stratum (as returned by `getStrataSplit`). |
| design | a character, logical or numeric vector specifying the variables (columns) to be used for stratification. |
| USE.NAMES | a logical indicating whether information about the strata should be used as `names` for the result. |
| split | an optional list in which each list element contains the indices of the observations belonging to the corresponding stratum (as returned by `getStrataSplit`). |

**Value**

For `getStrataLegend`, a `data.frame` describing the strata.

For `getStrataSplit`, a list in which each element contains the indices of the observations belonging to the corresponding stratum.

For `getStrataTable`, a `data.frame` describing the strata and containing the stratum sizes.

For `getStratumSizes`, a numeric vector of the stratum sizes.

For `getStratumValues`, a numeric vector giving the stratum number for each observation.

**Methods for function getStrataLegend**

**x = "data.frame", design = "BasicVector"** get a `data.frame` describing the strata, according to the variables specified by `design`.

**Methods for function getStrataSplit**

**x = "data.frame", design = "BasicVector"** get a list in which each element contains the indices of the observations belonging to the corresponding stratum, according to the variables specified by `design`.

**Methods for function getStrataTable**

**x = "data.frame", design = "BasicVector"** get a `data.frame` describing the strata and containing the stratum sizes, according to the variables specified by `design`.

**Methods for function getStratumSizes**

**x = "list", design = "missing"** get the stratum sizes for a list in which each list element contains the indices of the observations belonging to the corresponding stratum (as returned by `getStrataSplit`).

**x = "data.frame", design = "BasicVector"** get the stratum sizes of a data set, according to the variables specified by `design`.

**Methods for function getStratumValues**

**x = "data.frame", design = "BasicVector", split = "list"** get the stratum number for each obser-
vation, according to the variables specified by `design`. A previously computed list in which
each list element contains the indices of the observations belonging to the corresponding stra-
tum (as returned by `getStrataSplit`) speeds things up a bit.

**x = "data.frame", design = "BasicVector", split = "missing"** get the stratum number for each
observation, according to the variables specified by `design`.

### Author(s)

Andreas Alfons

### See Also

[stratify](), [Strata]()

### Examples

```
data(eusilcP)

## all data
getStrataLegend(eusilcP, c("region", "gender"))
getStrataTable(eusilcP, c("region", "gender"))
getStratumSizes(eusilcP, c("region", "gender"))

## small sample
sam <- draw(eusilcP, size = 25)
getStrataSplit(sam, "gender")
getStratumValues(sam, "gender")
```

---

| summary-methods | *Methods for producing a summary of an object* |
|---|---|

---

### Description

Produce a summary an object.

### Usage

```
## S4 method for signature 'SampleSetup'
summary(object)

## S4 method for signature 'SimControl'
summary(object)

## S4 method for signature 'SimResults'
summary(object, ...)
```

```
## S4 method for signature 'Strata'
summary(object)

## S4 method for signature 'VirtualContControl'
summary(object)

## S4 method for signature 'VirtualDataControl'
summary(object)

## S4 method for signature 'VirtualNAControl'
summary(object)

## S4 method for signature 'VirtualSampleControl'
summary(object)
```

## Arguments

object          an object.

...             additional arguments to be passed down to methods.

## Value

The form of the resulting object depends on the class of the argument object. See the "Methods" section below for details.

## Methods

signature(x = "SampleSetup") returns an object of class SummarySampleSetup, which contains information on the size of each of the set up samples.

signature(x = "SimControl") currently returns the object itself.

signature(x = "SimResults") produces a summary of the simulation results by calling the method of summary for the data.frame in slot values.

signature(x = "Strata") returns a data.frame containing the size of each stratum.

signature(x = "VirtualContControl") currently returns the object itself.

signature(x = "VirtualDataControl") currently returns the object itself.

signature(x = "VirtualNAControl") currently returns the object itself.

signature(x = "VirtualSampleControl") currently returns the object itself.

## Author(s)

Andreas Alfons

## References

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

## See Also

summary, "SampleSetup", "SummarySampleSetup", "SimResults", "Strata"

## Examples

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)

## class "Strata"
# set up samples using group sampling
strata <- stratify(eusilcP, "region")
summary(strata)
```

---

SummarySampleSetup-class

*Class "SummarySampleSetup"*

---

## Description

Class containing a summary of set up samples.

## Objects from the Class

Objects can be created by calls of the form new("SummarySampleSetup", ...) or SummarySampleSetup(...).

However, objects are expected to be created by the summary method for class "SampleSetup", these constructor functions are not supposed to be called by the user.

## Slots

size: Object of class "numeric" giving the size of each of the set up samples.

## Accessor methods

getSize signature(x = "SummarySampleSetup"): get slot size.

## Methods

show signature(object = "SummarySampleSetup"): print a summary of set up samples on the R console.

## UML class diagram

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

## Note

There are no mutator methods available since the slots are not supposed to be changed by the user.

## Author(s)

Andreas Alfons

## See Also

"SampleSetup", summary

## Examples

```
showClass("SummarySampleSetup")
```

---

| tail-methods | *Methods for returning the last parts of an object* |
|---|---|

---

## Description

Return the last parts of an object.

## Usage

```
## S4 method for signature 'SampleSetup'
tail(x, k = 6, n = 6, ...)

## S4 method for signature 'SimControl'
tail(x)

## S4 method for signature 'SimResults'
tail(x, ...)

## S4 method for signature 'Strata'
tail(x, ...)

## S4 method for signature 'VirtualContControl'
tail(x)

## S4 method for signature 'VirtualDataControl'
tail(x)
```

```
## S4 method for signature 'VirtualNAControl'
tail(x)

## S4 method for signature 'VirtualSampleControl'
tail(x)
```

### Arguments

| | |
|---|---|
| x | an object. |
| k | for objects of class "SampleSetup", the number of set up samples to be kept in the resulting object. |
| n | for objects of class "SampleSetup", the number of indices to be kept in each of the set up samples in the resulting object. |
| ... | additional arguments to be passed down to methods. |

### Value

An object of the same class as x, but in general smaller. See the "Methods" section below for details.

### Methods

signature(x = "SampleSetup") returns the last parts of set up samples. The last n indices of each of the last k set up samples are kept.

signature(x = "SimControl") currently returns the object itself.

signature(x = "SimResults") returns the last parts of simulation results. The method of [tail](#) for the data.frame in slot values is thereby called.

signature(x = "Strata") returns the last parts of strata information. The method of [tail](#) for the vector in slot values is thereby called and the slots split and size are adapted accordingly.

signature(x = "VirtualContControl") currently returns the object itself.

signature(x = "VirtualDataControl") currently returns the object itself.

signature(x = "VirtualNAControl") currently returns the object itself.

signature(x = "VirtualSampleControl") currently returns the object itself.

### Author(s)

Andreas Alfons

### References

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

### See Also

[tail](#), ["SampleSetup"](#), ["SimResults"](#), ["Strata"](#)

## Examples

```
## load data
data(eusilcP)

## class "SampleSetup"
# set up samples using group sampling
set <- setup(eusilcP, grouping = "hid", size = 1000, k = 50)
summary(set)
# get the last 10 indices of each of the last 5 samples
tail(set, k = 5, n = 10)

## class "Strata"
# set up samples using group sampling
strata <- stratify(eusilcP, "region")
summary(strata)
# get strata information for the last 10 observations
tail(strata, 10)
```

---

TwoStageControl-class    *Class "TwoStageControl"*

---

## Description

Class for controlling the setup of samples using a two-stage procedure.

## Usage

```
TwoStageControl(..., fun1 = srs, fun2 = srs, size1 = NULL,
                size2 = NULL, prob1 = NULL, prob2 = NULL,
                dots1 = list(), dots2 = list())
```

## Arguments

| | |
|---|---|
| ... | the slots for the new object (see below). |
| fun1 | the function to be used for sampling in the first stage (the first list component of slot fun). |
| fun2 | the function to be used for sampling in the second stage (the second list component of slot fun). |
| size1 | the number of PSUs to sample in the first stage (the first list component of slot size). |
| size2 | the number of items to sample in the second stage (the second list component of slot size). |
| prob1 | the probability weights for the first stage (the first list component of slot prob). |
| prob2 | the probability weights for the second stage (the second list component of slot prob). |

| dots1 | additional arguments to be passed to the function for sampling in the first stage (the first list component of slot `dots`). |
| dots2 | additional arguments to be passed to the function for sampling in the second stage (the second list component of slot `dots`). |

### Objects from the Class

Objects can be created by calls of the form `new("TwoStageControl", ...)` or via the constructor `TwoStageControl`.

### Slots

`design`: Object of class `"BasicVector"` specifying variables (columns) to be used for stratified sampling in the first stage.

`grouping`: Object of class `"BasicVector"` specifying grouping variables (columns) to be used for sampling primary sampling units (PSUs) and secondary sampling units (SSUs), respectively.

`fun`: Object of class `"list"`; a list of length two containing the functions to be used for sampling in the first and second stage, respectively (defaults to [srs](#) for both stages). The functions should return a vector containing the indices of the sampled items.

`size`: Object of class `"list"`; a list of length two, where each component contains an optional non-negative integer giving the number of items to sample in the first and second stage, respectively. In case of stratified sampling in the first stage, a vector of non-negative integers, each giving the number of PSUs to sample from the corresponding stratum, may be supplied. For the second stage, a vector of non-negative integers giving the number of items to sample from each PSU may be used.

`prob`: Object of class `"list"`; a list of length two, where each component gives optional probability weights for the first and second stage, respectively. Each component may thereby be a numerical vector, or a character string or integer vector specifying a variable (column) that contains the probability weights.

`dots`: Object of class `"list"`; a list of length two, where each component is again a list containing additional arguments to be passed to the corresponding function for sampling in `fun`.

`k`: Object of class `"numeric"`; a single positive integer giving the number of samples to be set up.

### Details

There are some restrictions on the argument names of the functions for sampling in `fun`. If the sampling method needs population data as input, the corresponding argument should be called `x` and should expect a `data.frame`. If it only needs the population size as input, the argument should be called `N`. Note that the function is not expected to have both `x` and `N` as arguments, and that the latter is typically much faster. Furthermore, if the function has arguments for sample size and probability weights, they should be called `size` and `prob`, respectively. Note that a function with `prob` as its only argument is perfectly valid (for probability proportional to size sampling). Further arguments may be supplied as a list via the slot `dots`.

### Extends

Class "[VirtualSampleControl](#)", directly. Class "[OptSampleControl](#)", by class "VirtualSample-Control", distance 2.

**Accessor and mutator methods**

In addition to the accessor and mutator methods for the slots inherited from *"VirtualSampleControl"*, the following are available:

getDesign signature(x = "TwoStageControl"): get slot design.

setDesign signature(x = "TwoStageControl"): set slot design.

getGrouping signature(x = "TwoStageControl"): get slot grouping.

setGrouping signature(x = "TwoStageControl"): set slot grouping.

getCollect signature(x = "TwoStageControl"): get slot collect.

setCollect signature(x = "TwoStageControl"): set slot collect.

getFun signature(x = "TwoStageControl"): get slot fun.

setFun signature(x = "TwoStageControl"): set slot fun.

getSize signature(x = "TwoStageControl"): get slot size.

setSize signature(x = "TwoStageControl"): set slot size.

getProb signature(x = "TwoStageControl"): get slot prob.

setProb signature(x = "TwoStageControl"): set slot prob.

getDots signature(x = "TwoStageControl"): get slot dots.

setDots signature(x = "TwoStageControl"): set slot dots.

**Methods**

In addition to the methods inherited from *"VirtualSampleControl"*, the following are available:

clusterSetup signature(cl = "ANY", x = "data.frame",    control = "TwoStageControl"): set up multiple samples on a cluster.

setup signature(x = "data.frame",    control = "TwoStageControl"): set up multiple samples.

show signature(object = "TwoStageControl"): print the object on the R console.

**UML class diagram**

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

**Author(s)**

Andreas Alfons

**See Also**

*"VirtualSampleControl"*, *"SampleControl"*, *"SampleSetup"*, setup, draw

**Examples**

```
showClass("TwoStageControl")
```

---

VirtualContControl-class

*Class "VirtualContControl"*

---

**Description**

Virtual superclass for controlling contamination in a simulation experiment.

**Objects from the Class**

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

**Slots**

target: Object of class "OptCharacter"; a character vector specifying specifying the variables (columns) to be contaminated, or NULL to contaminate all variables (except the additional ones generated internally).

epsilon: Object of class "numeric" giving the contamination levels.

**Extends**

Class "OptContControl", directly.

**Accessor and mutator methods**

getTarget signature(x = "VirtualContControl"): get slot target.

setTarget signature(x = "VirtualContControl"): set slot target.

getEpsilon signature(x = "VirtualContControl"): get slot epsilon.

setEpsilon signature(x = "VirtualContControl"): set slot epsilon.

**Methods**

head signature(x = "VirtualContControl"): currently returns the object itself.

length signature(x = "VirtualContControl"): get the number of contamination levels to be used.

show signature(object = "VirtualContControl"): print the object on the R console.

summary signature(object = "VirtualContControl"): currently returns the object itself.

tail signature(x = "VirtualContControl"): currently returns the object itself.

**UML class diagram**

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

#### Author(s)

Andreas Alfons

#### References

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

#### See Also

"DCARContControl", "DARContControl", "ContControl", contaminate

#### Examples

```
showClass("VirtualContControl")
```

---

VirtualDataControl-class
                                *Class "VirtualDataControl"*

---

#### Description

Virtual superclass for controlling model-based generation of data.

#### Objects from the Class

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

#### Extends

Class "OptDataControl", directly.

#### Methods

clusterRunSimulation signature(cl = "ANY",      x = "VirtualDataControl", setup = "missing", nrep = "nu
    run a simulation experiment on a cluster.

clusterRunSimulation signature(cl = "ANY",      x = "VirtualDataControl", setup = "VirtualSampleContro
    run a simulation experiment on a cluster.

head signature(x = "VirtualContControl"): currently returns the object itself.

runSimulation signature(x = "VirtualDataControl",      setup = "missing", nrep = "numeric", control = "
    run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",      setup = "missing", nrep = "missing", control = "
    run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",      setup = "VirtualSampleControl", nrep = "numeric
    run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",      setup = "VirtualSampleControl", nrep = "missing
    run a simulation experiment.

summary signature(object = "VirtualContControl"): currently returns the object itself.

tail signature(x = "VirtualContControl"): currently returns the object itself.

### UML class diagram

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

### Author(s)

Andreas Alfons

### References

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

### See Also

"DataControl", generate

### Examples

showClass("VirtualDataControl")

---

VirtualNAControl-class
                              *Class "VirtualNAControl"*

---

### Description

Virtual superclass for controlling the insertion of missing values in a simulation experiment.

### Objects from the Class

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

## Slots

target: Object of class ″OptCharacter″; a character vector specifying the variables (columns) in which missing values should be inserted, or NULL to insert missing values in all variables (except the additional ones generated internally).

NArate: Object of class ″NumericMatrix″ giving the missing value rates, which may be selected individually for the target variables. In case of a vector, the same missing value rates are used for all target variables. In case of a matrix, on the other hand, the missing value rates to be used for each target variable are given by the respective column.

## Extends

Class ″OptNAControl″, directly.

## Accessor and mutator methods

getTarget signature(x = ″VirtualNAControl″): get slot target.

setTarget signature(x = ″VirtualNAControl″): set slot target.

getNArate signature(x = ″VirtualNAControl″): get slot NArate.

setNArate signature(x = ″VirtualNAControl″): set slot NArate.

## Methods

head signature(x = ″VirtualNAControl″): currently returns the object itself.

length signature(x = ″VirtualNAControl″): get the number of missing value rates to be used (the length in case of a vector or the number of rows in case of a matrix).

show signature(object = ″VirtualNAControl″): print the object on the R console.

summary signature(object = ″VirtualNAControl″): currently returns the object itself.

tail signature(x = ″VirtualNAControl″): currently returns the object itself.

## UML class diagram

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

## Author(s)

Andreas Alfons

## References

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

## See Also

″NAControl″, setNA

### Examples

```
showClass("VirtualNAControl")
```

---

VirtualSampleControl-class

*Class "VirtualSampleControl"*

---

### Description

Virtual superclass for controlling the setup of samples.

### Objects from the Class

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

### Slots

k: Object of class "numeric", a single positive integer giving the number of samples to be set up.

### Extends

Class "OptSampleControl", directly.

### Accessor and mutator methods

getK signature(x = "VirtualSampleControl"): get slot k.

setK signature(x = "VirtualSampleControl"): set slot k.

### Methods

clusterRunSimulation signature(cl = "ANY",   x = "data.frame", setup = "VirtualSampleControl", nrep
     run a simulation experiment on a cluster.

clusterRunSimulation signature(cl = "ANY",   x = "VirtualDataControl", setup = "VirtualSampleContro
     run a simulation experiment on a cluster.

draw signature(x = "data.frame",    setup = "VirtualSampleControl"): draw a
     sample.

head signature(x = "VirtualSampleControl"): currently returns the object itself.

length signature(x = "VirtualSampleControl"): get the number of samples to be set up.

runSimulation signature(x = "data.frame",   setup = "VirtualSampleControl", nrep = "missing",    c
     run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",   setup = "VirtualSampleControl", nrep = "numeric
     run a simulation experiment.

runSimulation signature(x = "VirtualDataControl",   setup = "VirtualSampleControl", nrep = "missing
     run a simulation experiment.

show signature(object = "VirtualSampleControl"): print the object on the R console.

summary signature(object = "VirtualSampleControl"): currently returns the object itself.

tail signature(x = "VirtualSampleControl"): currently returns the object itself.

## UML class diagram

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

## Author(s)

Andreas Alfons

## References

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

## See Also

"SampleControl", "TwoStageControl", "SampleSetup", setup, draw

## Examples

```
showClass("VirtualSampleControl")
```

# Index