Package 'saeSim'

October 14, 2022

Type Package

Title Simulation Tools for Small Area Estimation

Version 0.11.0

URL https://wahani.github.io/saeSim/

BugReports https://github.com/wahani/saeSim/issues

Depends R(>= 3.1), methods

Imports dplyr (>= 0.2), functional, ggplot2, grDevices, MASS, utils, spdep, stats, parallelMap, tibble

Suggests testthat, knitr, rmarkdown, covr

Description Tools for the simulation of data in the context of small area estimation. Combine all steps of your simulation - from data generation over drawing samples to model fitting - in one object. This enables easy modification and combination of different scenarios. You can store your results in a folder or start the simulation in parallel.

VignetteBuilder knitr

RoxygenNote 7.1.1

Encoding UTF-8

NeedsCompilation no

Author Sebastian Warnholz [aut, cre],
Timo Schmid [aut]

Maintainer Sebastian Warnholz <wahani@gmail.com>

Repository CRAN

Date/Publication 2022-02-07 16:40:02 UTC

R topics documented:

agg_all 2
as.data.frame.sim_setup 3

autoplot.sim_setup . . . 3
base_add_id . . . 4
base_id . . . 4
comp_var . . . 5
gen_norm . . . 6
plot.sim_setup . . . 7
sample_fraction . . . 8
show,sim_setup-method . . . 9
sim . . . 9
sim_agg . . . 11
sim_base . . . 12
sim_base_lm . . . 12
sim_comp_n . . . 13
sim_comp_pop . . . 14
sim_gen . . . 15
sim_gen_cont . . . 16
sim_gen_x . . . 17
sim_read_data . . . 18
sim_resp . . . 19
sim_sample . . . 20
sim_simName . . . 21
summary,sim_setup-method . . . 21
%>% . . . 22

Index **23**

`agg_all` *Aggregation function*

Description

This function is intended to be used with `sim_agg` and not interactively. This is one implementation for aggregating data in a simulation set-up.

Usage

```
agg_all(groupVars = "idD")
```

Arguments

`groupVars`	variable names as character identifying groups to be aggregated.

Details

This function follows the split-apply-combine idiom. Each data set is split by the defined variables. Then the variables within each subset are aggregated (reduced to one row). Logical variables are reduced by `any`; for characters and factors dummy variables are created and the aggregate is the mean of each dummy; and for numerics the mean (removing NAs).

See Also

`sim_agg`

Examples

```
sim_base() %>% sim_gen_x() %>% sim_gen_e() %>% sim_agg(agg_all())
```

`as.data.frame.sim_setup` *as.data.frame method for sim_setup*

Description

Use this method to get a single simulated data.frame out of a sim_setup object.

Usage

```
## S3 method for class 'sim_setup'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	a sim_setup
`row.names`	will have no effect
`optional`	will have no effect
`...`	will have no effect

`autoplot.sim_setup` *Autoplot method*

Description

Use this function to produce plots for an object of class `sim_setup` and you like to have plots based on ggplot2. At this time it is a ggplot2 implementation which mimics the behavior of `smoothScatter` without all the options.

Usage

```
## S3 method for class 'sim_setup'
autoplot(object, x = "x", y = "y", ...)
```

Arguments

`object`	a sim_setup
`x`	character of variable name in the data on the x-axis
`y`	character of variable name in the data on the y-axis
`...`	is not used

Examples

```
## Not run:
autoplot(sim_base_lm())

## End(Not run)
```

`base_add_id` *Add id-variables to data*

Description

Use this function to add id-variables to your data.

Usage

```
base_add_id(data, domainId)
```

Arguments

`data`	a data.frame.
`domainId`	variable names in `data` as character which will identify the areas/domains/groups/cluster in the data.

`base_id` *Construct data with id-variables*

Description

This function constructs a data frame with grouping/`id` variables.

Usage

```
base_id(nDomains = 10, nUnits = 10)

base_id_temporal(nDomains = 10, nUnits = 10, nTime = 10)
```

Arguments

`nDomains`	The number of domains.
`nUnits`	The number of units in each domain. Can have `length(nUnits) > 1`.
`nTime`	The number of time points for each units.

Value

Return a `data.frame` with variables `idD` as ID-variable for domains, and `idU` as ID-variable for units.

Examples

```
base_id(2, 2)
base_id(2, c(2, 3))
```

`comp_var` *Compute variables in data*

Description

This function is intended to be used with `sim_comp_pop`, `sim_comp_sample` or `sim_comp_agg` and not interactively. This is a wrapper around `mutate`

Usage

```
comp_var(...)
```

Arguments

`...`	variables interpreted in the context of that data frame.

See Also

`sim_comp_pop`, `sim_comp_sample`, `sim_comp_agg`

Examples

```
sim_base_lm() %>% sim_comp_pop(comp_var(yExp = exp(y)))
```

`gen_norm` *Generator functions*

Description

These functions are intended to be used with `sim_gen` and not interactively. They are designed to draw random numbers according to the setting of grouping variables.

Usage

```
gen_norm(mean = 0, sd = 1, name = "e")

gen_v_norm(mean = 0, sd = 1, name = "v")

gen_v_sar(mean = 0, sd = 1, rho = 0.5, type = "rook", name)

gen_v_ar1(mean = 0, sd = 1, rho = 0.5, groupVar = "idD", timeVar = "idT", name)

gen_generic(generator, ..., groupVars = NULL, name)
```

Arguments

`mean`	the mean passed to the random number generator, for example `rnorm`.
`sd`	the standard deviation passed to the random number generator, for example rnorm.
`name`	name of variable as character in which random numbers are stored.
`rho`	the correlation used to create the variance covariance matrix for a SAR process - see `cell2nb`.
`type`	either "rook" or "queen". See `cell2nb` for details.
`groupVar`	a variable name identifying groups.
`timeVar`	a variable name identifying repeated measurements.
`generator`	a function producing random numbers.
`...`	arguments passed to `generator`.
`groupVars`	names of variables as character. Identify groups within random numbers are constant.

Details

`gen_norm` is used to draw random numbers from a normal distribution where all generated numbers are independent.

`gen_v_norm` and `gen_v_sar` will create an area-level random component. In the case of `v_norm`, the error component will be from a normal distribution and i.i.d. from an area-level perspective (all units in an area will have the same value, all areas are independent). v_sar will also be from a normal distribution, but the errors are correlated. The variance covariance matrix is constructed for a SAR(1) - spatial/simultanous autoregressive process. mvrnorm is used for the random number generation. `gen_v_norm` and `gen_v_sar` expect a variable `idD` in the data identifying the areas.

`gen_generic` can be used if your world is not normal. You can specify ’any’ function as generator, like `rnorm`. Arguments in `...` are matched by name or position. The first argument of `generator` is expected to be the number of random numbers (not necessarily named `n`) and need not to be specified.

See Also

`sim_gen`, `sim_gen_x`, `sim_gen_e`, `sim_gen_ec`, `sim_gen_v`, `sim_gen_vc`, `cell2nb`

Examples

```
sim_base() %>% sim_gen_x() %>% sim_gen_e() %>% sim_gen_v() %>% sim_gen(gen_v_sar(name = ”vSP”))

# Generic interface
set.seed(1)
dat1 <- sim(base_id() %>%
  sim_gen(gen_generic(rnorm, mean = 0, sd = 4, name = ”e”)))
set.seed(1)
dat2 <- sim(base_id() %>% sim_gen_e())
all.equal(dat1, dat2)
```

`plot.sim_setup` *Plotting methods*

Description

Use this function to produce plots for an object of class `sim_setup`.

Usage

```
## S3 method for class 'sim_setup'
plot(x, y, ...)
```

Arguments

`x`	a `sim_setup`
`y`	will be ignored
`...`	Arguments to be passed to `plot`.

See Also

`autoplot`

`sample_fraction` *Sampling functions*

Description

These functions are intended to be used with `sim_sample` and not interactively. They are wrappers around sample_frac and sample_n.

Usage

```
sample_fraction(size, replace = FALSE, weight = NULL, groupVars = NULL)

sample_number(size, replace = FALSE, weight = NULL, groupVars = NULL)

sample_numbers(size, replace = FALSE, groupVars = NULL)

sample_cluster_number(size, replace = FALSE, weight = NULL, groupVars)

sample_cluster_fraction(size, replace = FALSE, weight = NULL, groupVars)
```

Arguments

`size`	<`tidy-select`> For `sample_n()`, the number of rows to select. For `sample_frac()`, the fraction of rows to select. If `tbl` is grouped, `size` applies to each group.
`replace`	Sample with or without replacement?
`weight`	<`tidy-select`> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
`groupVars`	character with names of variables to be used for grouping.

Details

`sample_numbers` is a vectorized version of `sample_number`.

`sample_cluster_number` and `sample_cluster_fraction` will sample clusters (all units in a cluster).

Examples

```
sim_base_lm() %>% sim_sample(sample_number(5))
sim_base_lm() %>% sim_sample(sample_fraction(0.5))
sim_base_lm() %>% sim_sample(sample_cluster_number(5, groupVars = "idD"))
sim_base_lm() %>% sim_sample(sample_cluster_fraction(0.5, groupVars = "idD"))
```

`show,sim_setup-method` *Show for* `sim_setup`

Description

This is the documentation for the show methods in the package `saeSim`. In case you don't know, `show` is for S4-classes like `print` for S3. If you don't know what that means, don't bother, there is no reason to call `show` directly, however there is the need to document it.

Usage

```
## S4 method for signature 'sim_setup'
show(object)

## S4 method for signature 'summary.sim_setup'
show(object)
```

Arguments

`object`	Any R object

Details

Will print the head of a `sim_setup` to the console, after converting it to a `data.frame`.

`sim` *Start simulation*

Description

This function will start the simulation. Use the printing method as long as you are testing the scenario.

Usage

```
sim(
  x,
  R = 1,
  path = NULL,
  overwrite = TRUE,
  ...,
  suffix = NULL,
  fileExt = ".csv",
  libs = NULL,
  exports = NULL
)
```

Arguments

`x`	a `sim_setup`
`R`	number of repetitions.
`path`	optional path in which the simulation results can be saved. They will we coerced to a `data.frame` and then saved as 'csv'.
`overwrite`	`TRUE/FALSE`. If `TRUE` files in `path` are replaced. If `FALSE` files in `path` are not replaced and simulation will not be recomputed.
`...`	arguments passed to `parallelStart`.
`suffix`	an optional suffix of file names.
`fileExt`	the file extension. Default is ".csv" - alternative it can be ".RData".
`libs`	arguments passed to `parallelLibrary`. Will be used in a call to `do.call` after coersion with `as.list`.
`exports`	arguments passed to `parallelExport`. Will be used in a call to `do.call` after coersion with `as.list`.

Details

The package parallelMap is utilized as back-end for parallel computations.

Use the argument `path` to store the simulation results in a directory. This may be a good idea for long running simulations and for those using large `data.frames`. You can use `sim_read_data` to read them in. The return value will change to NULL in each run.

Value

The return value is a list. The elements are the results of each simulation run, typically of class `data.frame`. In case you specified `path`, each element is `NULL`.

Examples

```
setup <- sim_base_lm()
resultList <- sim(setup, R = 1)

# For parallel computations you may need to export objects
localFun <- function() cat("Hello World!")
comp_fun <- function(dat) {
  localFun()
  dat
}

res <- sim_base_lm() %>%
  sim_comp_pop(comp_fun) %>%
  sim(R = 2,
      mode = "socket", cpus = 2,
      exports = "localFun")

str(res)
```

`sim_agg` *Aggregation component*

Description

One of the components which can be added to a simulation set-up. Aggregating the data is a simulation component which can be used to aggregate the population or sample. The aggregation will simply be done after the sampling, if you haven't specified any sampling component, the population is aggregated (makes sense if you draw samples directly from the model).

Usage

```
sim_agg(simSetup, aggFun = agg_all())
```

Arguments

`simSetup`	a `sim_setup`.
`aggFun`	function which controls the aggregation process. At the moment only `agg_all` is defined.

Details

Potentially you can define an `aggFun` yourself. Take care that it only has one argument, named `dat`, and returns the aggregated data as `data.frame`.

See Also

`agg_all`, `sim_gen`, `sim_comp_pop`, `sim_sample`, , `sim_comp_sample`

Examples

```
# Aggregating the population:
sim_base_lm() %>% sim_agg()

# Aggregating after sampling:
sim_base_lm() %>% sim_sample() %>% sim_agg()

# User aggFun:
sim_base_lm() %>% sim_agg(function(dat) dat[1, ])
```

`sim_base` *Base component*

Description

Use the ‘sim_base‘ functions to start a new `sim_setup`.

Usage

```
sim_base(data = base_id(100, 100))
```

Arguments

`data`	a `data.frame`.

Examples

```
# Example for a linear model:
sim_base() %>% sim_gen_x() %>% sim_gen_e()
```

`sim_base_lm` *Preconfigured set-ups*

Description

`sim_base_lm()` will start a linear model: One regressor, one error component. `sim_base_lmm()` will start a linear mixed model: One regressor, one error component and one random effect for the domain. `sim_base_lmc()` and `sim_base_lmmc()` add outlier contamination to the scenarios. Use these as a quick start, then you probably want to configure your own scenario.

Usage

```
sim_base_lm()

sim_base_lmm()

sim_base_lmc()

sim_base_lmmc()
```

Details

Additional information on the generated variables:

- nDomains: 100 domains
- nUnits: 100 in each domain
- x: is normally distributed with mean of 0 and sd of 4
- e: is normally distributed with mean of 0 and sd of 4
- v: is normally distributed with mean of 0 and sd of 1, it is a constant within domains
- e-cont: as e; probability of unit to be contaminated is 0.05; sd is then 150
- v-cont: as v; probability of area to be contaminated is 0.05; sd is then 40
- y = 100 + x + v + e

Examples

```
# The preconfigured set-ups:
sim_base_lm()
sim_base_lmm()
sim_base_lmc()
sim_base_lmmc()
```

`sim_comp_n` *Preconfigured computation components*

Description

`sim_comp_n` and `sim_comp_N` will add the sample and population size in each domain respectively. `sim_comp_popMean` and `sim_comp_popVar` the population mean and variance of the variable `y`. The data is expected to have a variable `idD` identifying domains.

Usage

```
sim_comp_n(simSetup)

sim_comp_N(simSetup)

sim_comp_popMean(simSetup)

sim_comp_popVar(simSetup)
```

Arguments

`simSetup`	a `sim_setup`.

`sim_comp_pop` *Calculation component*

Description

One of the components which can be added to a `sim_setup`. These functions can be used for adding new variables to the data.

Usage

```
sim_comp_pop(simSetup, fun = comp_var(), by = "")

sim_comp_sample(simSetup, fun = comp_var(), by = "")

sim_comp_agg(simSetup, fun = comp_var(), by = "")
```

Arguments

`simSetup`	a `sim_setup`.
`fun`	a function, see details.
`by`	names of variables as character; identifying groups for which fun is applied.

Details

Potentially you can define a function for computation yourself. Take care that it only has one argument, named `dat`, and returns a `data.frame`. Use `comp_var` for simple data manipulation. Functions added with `sim_comp_pop` are applied before sampling; `sim_comp_sample` after sampling. Functions added with `sim_comp_agg` after aggregation.

See Also

`comp_var`, `sim_gen`, `sim_agg`, `sim_sample`, `sim_comp_N`, `sim_comp_n`, `sim_comp_popMean`, `sim_comp_popVar`

Examples

```
# Standard behavior
sim_base() %>% sim_gen_x() %>% sim_comp_N()

# Custom data modifications
## Add predicted values of a linear model
library(saeSim)

comp_lm <- function(dat) {
  dat$linearPredictor <- predict(lm(y ~ x, data = dat))
  dat
}

sim_base_lm() %>% sim_comp_pop(comp_lm)
```

```
# or if applied after sampling
sim_base_lm() %>% sim_sample() %>% sim_comp_pop(comp_lm)
```

---

sim_gen *Generation component*

Description

One of the components which can be added to a `sim_setup`.

Usage

```
sim_gen(simSetup, generator)

sim_gen_generic(simSetup, ...)
```

Arguments

`simSetup`	a `sim_setup`.
`generator`	generator function used to generate random numbers.
`...`	arguments passed to `gen_generic`.

Details

Potentially you can define a `generator` yourself. Take care that it has one argument, named `dat`, and returns a `data.frame`. `sim_gen_generic` is a shortcut to `gen_generic`.

See Also

`gen_norm`, `gen_v_norm`, `gen_v_sar`, `sim_agg`, , `sim_comp_pop`, `sim_sample`, `sim_gen_x`, `sim_gen_e`, `sim_gen_v`, `sim_gen_vc`, `sim_gen_ec`

Examples

```
# Data setup for a mixed model
sim_base() %>% sim_gen_x() %>% sim_gen_v() %>% sim_gen_e()
# Adding contamination in the model error
sim_base() %>% sim_gen_x() %>% sim_gen_v() %>% sim_gen_e() %>% sim_gen_ec()

# Simple user defined generator:
gen_myVar <- function(dat) {
  dat["myVar"] <- rnorm(nrow(dat))
  dat
}

sim_base() %>% sim_gen_x() %>% sim_gen(gen_myVar)

# And a chi-sq(5) distributed 'random-effect':
sim_base() %>% sim_gen_generic(rchisq, df = 5, groupVars = "idD", name = "re")
```

`sim_gen_cont` *Generation Component for contamination*

Description

One of the components which can be added to a `sim_setup`. It is applied after functions added with `sim_gen`.

Usage

```
sim_gen_cont(simSetup, generator, nCont, type, areaVar = NULL, fixed = TRUE)
```

Arguments

`simSetup`	a `sim_setup`.
`generator`	generator function used to generate random numbers.
`nCont`	gives the number of contaminated observations. Values between 0 and 1 will be treated as probability. If type is 'unit' and length is larger than 1, the expected length is the number of areas. If type is 'area' and length is larger than 1 the values are interpreted as area positions; i.e. `c(1, 3)` is interpreted as the first and 3rd area in the data is contaminated.
`type`	"unit" or "area" - unit- or area-level contamination.
`areaVar`	character with variable name(s) identifying areas.
`fixed`	TRUE fixes the observations which will be contaminated. FALSE will result in a random selection of observations or areas.

See Also

`sim_gen`

Examples

```
sim_base_lm() %>%
  sim_gen_cont(gen_norm(name = "e"), nCont = 0.05, type = "unit", areaVar = "idD") %>%
  as.data.frame
```

---

sim_gen_x *Preconfigured generation components*

Description

These are some preconfigured generation components and all wrappers around `sim_gen` and `sim_gen_cont`.

Usage

```
sim_gen_x(simSetup, mean = 0, sd = 4, name = "x")

sim_gen_e(simSetup, mean = 0, sd = 4, name = "e")

sim_gen_ec(
  simSetup,
  mean = 0,
  sd = 150,
  name = "e",
  nCont = 0.05,
  type = "unit",
  areaVar = "idD",
  fixed = TRUE
)

sim_gen_v(simSetup, mean = 0, sd = 1, name = "v")

sim_gen_vc(
  simSetup,
  mean = 0,
  sd = 40,
  name = "v",
  nCont = 0.05,
  type = "area",
  areaVar = "idD",
  fixed = TRUE
)
```

Arguments

`simSetup`	a `sim_setup`.
`mean`	the mean passed to the random number generator, for example `rnorm`.
`sd`	the standard deviation passed to the random number generator, for example rnorm.
`name`	name of variable as character in which random numbers are stored.

`nCont`	gives the number of contaminated observations. Values between 0 and 1 will be treated as probability. If type is 'unit' and length is larger than 1, the expected length is the number of areas. If type is 'area' and length is larger than 1 the values are interpreted as area positions; i.e. `c(1, 3)` is interpreted as the first and 3rd area in the data is contaminated.
`type`	"unit" or "area" - unit- or area-level contamination.
`areaVar`	character with variable name(s) identifying areas.
`fixed`	TRUE fixes the observations which will be contaminated. FALSE will result in a random selection of observations or areas.

Details

`x`: fixed-effect component; `e`: model-error; `ec`: contaminated model error; `v`: random-effect (error constant for each domain); `vc` contaminated random-effect. Note that for contamination you are expected to add both, a non-contaminated component and a contaminated component.

`sim_read_data` *Read in simulated data*

Description

Functions to read in simulation data from folder. Can be csv or RData files.

Usage

```
sim_read_data(path, ..., returnList = FALSE)

sim_clear_data(path, ...)

sim_read_list(path)

sim_clear_list(path)
```

Arguments

`path`	path to the files you want to read in.
`...`	arguments passed to `read.csv`
`returnList`	if `TRUE` a list containing the data.frames. Very much like the output of sim. If `FALSE` a single data.frame is returned, using `bind_rows`

`sim_resp` *Response component*

Description

One of the components which can be added to a `sim_setup`.

Usage

```
sim_resp(simSetup, respFun)

sim_resp_eq(simSetup, ...)
```

Arguments

`simSetup`	a `sim_setup`.
`respFun`	a function constructing the response variable
`...`	<`data-masking`> Name-value pairs. The name gives the name of the column in the output. The value can be: • A vector of length 1, which will be recycled to the correct length. • A vector the same length as the current group (or the whole data frame if ungrouped). • `NULL`, to remove the column. • A data frame or tibble, to create multiple columns in the output.

Details

Potentially you can define an `respFun` yourself. Take care that it only has one argument, named `dat`, and returns the a `data.frame`.

See Also

`agg_all`, `sim_gen`, `sim_comp_pop`, `sim_sample`, , `sim_comp_sample`

Examples

```
base_id() %>% sim_gen_x() %>% sim_gen_e() %>% sim_resp_eq(y = 100 + 2 * x + e)
```

`sim_sample` *Sampling component*

Description

One of the components which can be added to a `sim_setup`. This component can be used to add a sampling mechanism to the simulation set-up. A sample will be drawn after the population is generated (`sim_gen`) and variables on the population are computed (`sim_comp_pop`).

Usage

```
sim_sample(simSetup, smplFun = sample_number(size = 5L, groupVars = "idD"))
```

Arguments

`simSetup`	a `sim_setup`.
`smplFun`	function which controls the sampling process.

Details

Potentially you can define a `smplFun` yourself. Take care that it has one argument, named `dat` being the data as data.frame, and returns the sample as data.frame.

See Also

`sample_number`, `sample_fraction`

Examples

```
# Simple random sample - 5% sample:
sim_base_lm() %>% sim_sample(sample_fraction(0.05))

# Simple random sampling proportional to size - 5% in each domain:
sim_base_lm() %>% sim_sample(sample_fraction(0.05, groupVars = "idD"))

# User defined sampling function:
sample_mySampleFun <- function(dat) {
  dat[sample.int(nrow(dat), 10), ]
}

sim_base_lm() %>% sim_sample(sample_mySampleFun)
```

`sim_simName` *Add a name to a sim_setup*

Description

Use this function to add a name to a `sim_setup` in case you are simulating different scenarios. This name will be added if you use the function sim for simulation

Usage

```
sim_simName(simSetup, name)
```

Arguments

`simSetup`	a `sim_setup`.
`name`	a character

Examples

```
sim_base_lm() %>% sim_simName("newName")
```

`summary,sim_setup-method` *Summary for a sim_setup*

Description

Reports a summary of the simulation setup.

Usage

```
## S4 method for signature 'sim_setup'
summary(object, ...)
```

Arguments

`object`	a `sim_setup`.
`...`	has no effect.

Examples

```
summary(sim_base_lm())
```

`%>%` *Piping operator*

Description

This is the 'pipe operator' from the package 'magrittr'. Use it to chain all operations for the simulation together. See the original documentation for details: %>%.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	The value to be piped
`rhs`	A function or expression

Index

%>%, *22*, 22

agg_all, 2, *11*, *19*
any, *2*
as.data.frame.sim_setup, 3
as.list, *10*
autoplot, *7*
autoplot (autoplot.sim_setup), 3
autoplot.sim_setup, 3

base_add_id, 4
base_id, 4
base_id_temporal (base_id), 4
bind_rows, *18*

cell2nb, *6*, *7*
comp_var, 5, *14*

do.call, *10*

gen_generic, *15*
gen_generic (gen_norm), 6
gen_norm, 6, *15*
gen_v_ar1 (gen_norm), 6
gen_v_norm, *15*
gen_v_norm (gen_norm), 6
gen_v_sar, *15*
gen_v_sar (gen_norm), 6

mutate, *5*
mvrnorm, *7*

parallelExport, *10*
parallelLibrary, *10*
parallelStart, *10*
plot, *7*
plot.sim_setup, 7

read.csv, *18*
rnorm, *6*, *7*, *17*

sample_cluster_fraction (sample_fraction), 8
sample_cluster_number (sample_fraction), 8
sample_frac, *8*
sample_fraction, 8, *20*
sample_n, *8*
sample_number, *20*
sample_number (sample_fraction), 8
sample_numbers (sample_fraction), 8
show,sim_setup-method, 9
show,summary.sim_setup-method (show,sim_setup-method), 9
sim, 9, *21*
sim_agg, *2*, *3*, 11, *14*, *15*
sim_base, 12
sim_base_lm, 12
sim_base_lmc (sim_base_lm), 12
sim_base_lmm (sim_base_lm), 12
sim_base_lmmc (sim_base_lm), 12
sim_clear_data (sim_read_data), 18
sim_clear_list (sim_read_data), 18
sim_comp_agg, *5*
sim_comp_agg (sim_comp_pop), 14
sim_comp_N, *14*
sim_comp_N (sim_comp_n), 13
sim_comp_n, 13, *14*
sim_comp_pop, *5*, *11*, 14, *15*, *19*, *20*
sim_comp_popMean, *14*
sim_comp_popMean (sim_comp_n), 13
sim_comp_popVar, *14*
sim_comp_popVar (sim_comp_n), 13
sim_comp_sample, *5*, *11*, *19*
sim_comp_sample (sim_comp_pop), 14
sim_gen, *6*, *7*, *11*, *14*, 15, *16*, *17*, *19*, *20*
sim_gen_cont, 16, *17*
sim_gen_e, *7*, *15*
sim_gen_e (sim_gen_x), 17
sim_gen_ec, *7*, *15*

`sim_gen_ec` *(*`sim_gen_x`*)*, 17
`sim_gen_generic` *(*`sim_gen`*)*, 15
`sim_gen_v`, *7*, *15*
`sim_gen_v` *(*`sim_gen_x`*)*, 17
`sim_gen_vc`, *7*, *15*
`sim_gen_vc` *(*`sim_gen_x`*)*, 17
`sim_gen_x`, *7*, *15*, 17
`sim_read_data`, *10*, 18
`sim_read_list` *(*`sim_read_data`*)*, 18
`sim_resp`, 19
`sim_resp_eq` *(*`sim_resp`*)*, 19
`sim_sample`, *8*, *11*, *14*, *15*, *19*, 20
`sim_simName`, 21
`smoothScatter`, *3*
`summary,sim_setup-method`, 21