Package ‘carts’

November 13, 2025

Type Package

Title Simulation-Based Assessment of Covariate Adjustment in Randomized Trials

Version 0.1.0

Maintainer Benedikt Sommer <benediktsommer92@gmail.com>

Description Monte Carlo simulation framework for different randomized clinical trial designs with a special emphasis on estimators based on covariate adjustment. The package implements regression-based covariate adjustment (Rosenblum & van der Laan (2010) <doi:10.2202/1557-4679.1138>) and a one-step estimator (Van Lancker et al (2024) <doi:10.48550/arXiv.2404.11150>) for trials with continuous, binary and count outcomes. The estimation of the minimum sample-size required to reach a specified statistical power for a given estimator uses bisection to find an initial rough estimate, followed by stochastic approximation (Robbins-Monro (1951) <doi:10.1214/aoms/1177729586>) to improve the estimate, and finally, a grid search to refine the estimate in the neighborhood of the current best solution.

License Apache License (>= 2)

Depends R (>= 4.1), lava (>= 1.8.0)

Imports data.table (>= 1.14), logger (>= 0.2.2), methods, progressr, R6 (>= 2.5.1), survival, targeted (>= 0.6), rlang

LinkingTo Rcpp (>= 1.0.11), RcppArmadillo

Suggests future, tinytest, MASS, geepack, covr, pdftools, knitr, mets (>= 1.3.4), rmarkdown, magick, pwr, pwrss

BugReports https://github.com/NovoNordisk-OpenSource/carts/issues

URL https://novonordisk-opensource.github.io/carts/, https://github.com/NovoNordisk-OpenSource/carts

VignetteBuilder knitr

ByteCompile yes

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Benedikt Sommer [aut, cre],
Klaus K. Holst [aut],
Foroogh Shamsi [aut],
Novo Nordisk A/S [cph]

Repository CRAN

Date/Publication 2025-11-13 21:20:02 UTC

Contents

aggrsurv *Aggregate data in counting process format*

Description

Aggregate data in counting process format. The aggregation is done within subject only.

Usage

```
aggrsurv(
  data,
  breaks,
  entry = "entry",
  exit = "exit",
  status = "status",
  id = "id",
  names = c("episode", "entry", "exit", "events", "exposure"),
  ...
)
```

Arguments

`data`	data.frame
`breaks`	vector of time points
`entry`	name of entry date variable
`exit`	name exit date variable
`status`	censoring / event variable
`id`	id variable
`names`	character vector of names of new variables
`...`	additional arguments to lower level functions

Value

data.table

Examples

```
dat <- data.table::data.table(
  id = c(1, 1, 1, 1, 2, 2, 2),
  entry = as.Date(c(
    "2021-01-01", "2021-01-20", "2021-02-28", "2021-06-01",
    "2021-01-01", "2021-01-14", "2021-09-01"
 )),
  status = c(1, 1, 1, 1, 1, 1, 0),
  x = rnorm(7)
)
```

```
dat[, exit := data.table::shift(entry, 1, type="lead"), by=id]
dat[, exit := replace(exit, .N, as.Date("2021-12-31")),
     by = id]

res <- aggrsurv(dat,
  breaks = c(182),
  entry = "entry", exit = "exit", status = "status", id = "id"
)
print(res)
```

`append<-.list` *Assignment function to append values to existing list*

Description

Assignment function to append values to existing list

Usage

```
## S3 replacement method for class 'list'
append(x, name = NULL) <- value
```

Arguments

`x`	existing list
`name`	optional name of new element
`value`	new value to add to existing list

Examples

```
x <- list()
append(x) <- 1
append(x, name = "a") <- 2
# duplicated names are allowed
append(x, name = "a") <- 3
x
```

`bisection` *Root finding by bisection*

Description

Root finding by bisection

Usage

```
bisection(f, interval, niter = 6, tol = 1e-12, verbose = TRUE, ...)
```

Arguments

`f`	function to find root of (monotonic)
`interval`	a vector containing the end-points of the interval to be searched for the root
`niter`	number of iterations
`tol`	stopping criterion (absolute difference in function evaluated at end points of current interval)
`verbose`	if TRUE additional messages are printed throughout the optimization
`...`	additional arguments passed to `f`

Value

numeric specifying the root

`covar_add` *Add additional covariates to existing list of covariates*

Description

For use with Trial objects, this function makes it possible to easily add additional covariates to an existing list of covariates (in the form of a data.frame or data.table).

Usage

```
covar_add(covars, x, names = NULL, ...)
```

Arguments

`covars`	list of covariates (data.frame's or data.table's)
`x`	new covariates (function or list of functions/scalars)
`names`	optional names of new covariates
`...`	additional arguments to function `x` or functions in `x`

Value

matching format of covariates in `covars`

Author(s)

Klaus Kähler Holst

Examples

```
# adding "fixed" treatment indicator in each period
n <- 5
xt <- function(n, ...) {
 covar_loggamma(n, normal.cor = 0.2) |>
   covar_add(list(a = 0, a = 1))
}
xt(n)
# adding randomized treatment indicator
xt <- function(n, ...) {
 covar_loggamma(n, normal.cor = 0.2) |>
   covar_add(list(a = rbinom(n, 1, 0.5), a = rbinom(n, 1, 0.5)))
}
xt(5)
# adding baseline covariates
xt <- function(n, ...) {
 covar_loggamma(n, normal.cor = 0.2) |>
   covar_add(rnorm(n), names = "w1") |> # data
   covar_add(list(w2 = rnorm(n))) |> # data
   covar_add(data.frame(w3 = rnorm(n))) |> # data
   covar_add(\(n) data.frame(w4 = rnorm(n))) |> # function
   covar_add(\(n) rnorm(n), names = "w5") # function
}
xt(5)
```

---

covar_bootstrap *Sample from empirical distribution of covariate data*

Description

Sample from empirical distribution of covariate data

Usage

```
covar_bootstrap(data, subset = NULL)
```

Arguments

data	data.frame
subset	optional columns to select from data frame

Value

random generator (function)

covar_join — *Add additional covariates to existing covariate random generator*

Description

For use with Trial objects, this function makes it possible to easily add additional covariates to an existing random generator (function(n ...) returning a data.frame or data.table)

Usage

```
covar_join(f, ...)
```

Arguments

`f`	covariate random generator
`...`	additional covariate generators or constant covariates

Value

function, with returned data type matching that of `f`

Examples

```
# single period
n <- 5
c1 <- function(n) data.frame(a = rnorm(n))
c2 <- function(n) data.frame(b = rnorm(n))
x <- c1 %join% c2
x(n)

# adding covariates that remain constant when sampling
x <- c1 %join% data.frame(b = rnorm(n))
all.equal(x(n)$b, x(n)$b)

# adding multiple anonymous functions require parenthesis enclosing, with
# the exception of the last function
x <- c1 %join%
 (\(n) data.frame(b = rnorm(n))) %join%
 \(n) data.frame(c = rnorm(n))
x(n)

# multiple periods
base <- setargs(covar_loggamma, normal.cor = .5)
x <- base %join%
  function(n) list(
      data.frame(a = rbinom(n, 1, 0.5)),
      data.frame(a = rbinom(n, 1, 0.5))
    )
x(n)
```

```
# constant covariate
x <- base %join% list(data.frame(a = 0), data.frame(a = 1))
x(n)

# baseline covariate
x <- base %join% function(n) data.frame(w = rnorm(n))
x(n)
```

---

covar_loggamma *Simulate from a log gamma-gaussian copula distribution*

Description

Simulate from the logarithmic transform of a Gaussian copula model with compound symmetry correlation structure and with Gamma distributed marginals with mean one.

Usage

```
covar_loggamma(
  n,
  normal.cor = NULL,
  gamma.var = 1,
  names = c("z"),
  type = "cs",
  ...
)
```

Arguments

n	Number of samples
normal.cor	Correlation parameter (n x r) or (1 x r) matrix
gamma.var	Variance of gamma distribution (n x p or 1 x p matrix)
names	Column name of the column vector (default "z")
type	of correlation matrix structure (cs: compound-symmetry / exchangable, ar: autoregressive, un: unstructured, to: toeplitz). The dimension of `normal.cor` must match, i.e., for a Toeplitz correlation matrix r = p-1, and for a cs and ar r=1.
...	Additional arguments passed to lower level functions

Details

We simulate from the Gaussian copula by first drawing $X \sim N(0, R)$ and transform the margins with $x \mapsto \log(F_\nu^{-1}\{\Phi(x)\})$ where Φ is the standard normal CDF and F_ν^{-1} is the quantile function of the Gamma distribution with scale and rate parameter equal to ν.

Value

list of data.tables

See Also

outcome_count Trial covar_normal

`covar_normal` *Simulate from multivariate normal distribution*

Description

Simulate from MVN with compound symmetry variance structure and mean zero. The result is returned as a list where the ith element is the column vector with n observations from the ith coordinate of the MVN.

Usage

```
covar_normal(
  n,
  normal.cor = NULL,
  normal.var = 1,
  names = c("z"),
  type = "cs",
  ...
)
```

Arguments

`n`	Number of samples
`normal.cor`	Correlation parameter (n x r) or (1 x r) matrix
`normal.var`	marginal variance (can be specified as a p-dim. vector or a nxp matrix)
`names`	Column name of the column vector (default "z")
`type`	of correlation matrix structure (cs: compound-symmetry / exchangable, ar: autoregressive, un: unstructured, to: toeplitz). The dimension of `normal.cor` must match, i.e., for a Toeplitz correlation matrix r = p-1, and for a cs and ar r=1.
`...`	Additional arguments passed to lower level functions

Value

list of data.tables

See Also

outcome_count Trial covar_loggamma

`derive_covar_distribution` *Derive covariate distribution from covariate data type*

Description

Derive covariate distribution (for outcome regression) for integers (Poisson), numeric (Gaussian) and binary (Binomial) data. Raise an error for other data types.

Usage

```
derive_covar_distribution(covar)
```

Arguments

`covar`	Vector with covariates

Value

lava package random generator function

Author(s)

Benedikt Sommer

`estimate_covar_model_full_cond` *Full conditional covariate simulation model*

Description

Estimates a full conditional model to approximate the joint distribution of covariate data. Each factor $p(x_i|x_1,\dots x_{i-1})$ is modelled with a `glm`, with mean $E[x_i|x_1,\dots x_{i-1}] = g^{-1}(\beta_0+\sum_{j=1}^{i-1}\beta_j x_j)$. The parametric distribution of each factor is either derived from the column type (see `derive_covar_distribution`) or specified by `cond.dist`.

Usage

```
estimate_covar_model_full_cond(data, cond.dist = NULL)
```

Arguments

`data`	Covariate `data.table`
`cond.dist`	`list` with random generator functions for the conditional distribution of each covariate

Value

lava::lvm object with estimated coefficients

Author(s)

Benedikt Sommer

Examples

```
data <- data.table::data.table(
y = as.factor(rbinom(1e3, size = 1, prob=0.1))
)

# infer distribution of y from column type
m.est <- estimate_covar_model_full_cond(data)
y <- sample_covar_parametric_model(1e4, m.est)$y |> as.integer() - 1
print(mean(y))

# specify distribution of y
m.est <- estimate_covar_model_full_cond(
  data, cond.dist = list(y = binomial.lvm)
)
y <- sample_covar_parametric_model(1e4, m.est)$y |> as.integer() - 1
print(mean(y))
```

`est_adj` *Construct estimator for the treatment effect in RCT based on covariate adjustment*

Description

Efficient estimator of the treatment effect based on the efficient influence function. This involves a model for the conditional mean of the outcome variable given covariates (Q-model). The implementation is a one-step estimator as described by Van Lancker et al (2024).

Usage

```
est_adj(
  response = "y",
  treatment = "a",
  covariates = NULL,
  offset = NULL,
  id = NULL,
  family = gaussian(),
  level = 0.95,
  treatment.effect = "absolute",
  nfolds = 1,
  ...
)
```

Arguments

`response`	(character, formula, targeted::learner) The default behavior when providing a character is to use a glm with treatment-covariate interactions for the Q-model. The covariates are specified via the `covariates` argument, where the default behavior is to use no covariates. When providing a formula, a glm is used for the Q-model, where the design matrix is specified by the formula. The last option is to provide a targeted::learner object that specifies the Q-model (see examples).
`treatment`	(character) Treatment variable. Additional care must be taken when the treatment variable is encoded as a factor (see examples).
`covariates`	(character) List of covariates. Only applicable when `response` is a character.
`offset`	(character) Optional offset to include in the glm model when `response` is a character.
`id`	(character) Subject id variable
`family`	(family) Family argument used in the glm when `response` is a character or formula.
`level`	(numeric) Confidence interval level
`treatment.effect`	(character, function) Default is the average treatment effect, i.e. difference in expected outcomes (x, y) -> x - y, with x = E[Y(1)] and y = E[Y(0)]). Other options are "logrr" (x, y) -> log(x / y)) and "logor" (x, y) -> log(x / (1 - x) * y / (1 - y)). A user-defined function can alternatively be provided to target a population parameter other than the absolute difference, log rate ratio or log odds ratio (see details).
`nfolds`	(integer) Number of folds for estimating the conditional average treatment effect with double machine learning.
`...`	Additional arguments to targeted::learner_glm when `response` is a character or formula.

Details

The user-defined function for `treatment.effect` needs to accept a single argument `x` of estimates of (E[Y(1)],E[Y(0)]). The estimates are a vector, where the order of E[Y(1)] and E[Y(0)] depends on the encoding of the `treatment` variable. E[Y(0)] is the first element when the treatment variable is drawn from a Bernoulli distribution and kept as a numeric variable or corresponds to the first level when the treatment variable is encoded as a factor.

Value

function

Author(s)

Klaus Kähler Holst

References

Van Lancker et al (2024) Automated, efficient and model-free inference for randomized clinical trials via data-driven covariate adjustment, arXiv:2404.11150

See Also

Trial est_glm

Examples

```
trial <- Trial$new(
    covariates = function(n) data.frame(a = rbinom(n, 1, 0.5), x = rnorm(n)),
    outcome = setargs(outcome_count,
      mean = ~ 1 + a*x,
      par = c(1, -0.1, 0.5, 0.2),
      overdispersion = 2)
)
dd <- trial$simulate(1e4)

# equivalent specifications to estimate log(E[Y(1)] / E[Y(0)])
estimators <- list(
  est_adj(family = poisson, treatment.effect = "logrr"),
  est_glm(family = poisson),
  est_adj(response = y ~ a, family = poisson, treatment.effect = "logrr"),
  est_adj(response = targeted::learner_glm(y ~ a, family = poisson),
    treatment.effect = "logrr"
  )
)
lapply(estimators, \(est) est(dd))


# now with covariates, estimating E[Y(1)] - E[Y(0)]
estimators <- list(
  est_adj(covariates = "x", family = poisson),
  est_adj(response = y ~ a * x, family = poisson),
  est_adj(response = targeted::learner_glm(y ~ a * x, family = poisson))
)
lapply(estimators, \(est) est(dd))

# custom treatment.effect function
estimator <- est_adj(response = y ~ a * x, family = poisson,
  treatment.effect = \(x) x[2] - x[1] # x[1] contains the estimate of E[Y(0)]
)
estimator(dd)

dd_factor <- dd
# when using factors, the control/comparator treatment needs to be the first
# level to estimate the contrasts defined by the `treatment.level` argument
estimator <- est_adj(response = y ~ a * x, family = poisson)
dd_factor$a <- factor(dd_factor$a, levels = c(0, 1))
estimator(dd_factor) # E[Y(1)] - E[Y(0)]
```

```
dd_factor$a <- factor(dd_factor$a, levels = c(1, 0))
estimator(dd_factor) # E[Y(1)] - E[Y(0)]
```

est_glm *Construct estimator for the treatment effect in RCT*

Description

Regression-based covariate adjustment as described by Rosenblum & van der Laan (2010). Standard errors are estimated with the Hubert-White sandwich estimator, instead using the efficient influence function as described in the paper. Available parametric models are (stats::glm and MASS::glm.nb).

Usage

```
est_glm(
  response = "y",
  treatment = "a",
  covariates = NULL,
  offset = NULL,
  id = NULL,
  level = 0.95,
  family = gaussian(),
  target.parameter = treatment,
  ...
)
```

Arguments

`response`	(character) Response variable
`treatment`	(character) Treatment variable. Additional care must be taken when the treatment variable is encoded as a factor (see examples).
`covariates`	(character; optional) Single or vector of covariates
`offset`	(character; optional) Model offset
`id`	(character; optional) Subject id variable
`level`	(numeric) Confidence interval level
`family`	(family or character) Exponential family that is supported by stats::glm and MASS::glm.nb
`target.parameter`	(character) Target parameter from model output
`...`	Additional arguments to lava::estimate

Value

function

Author(s)

Klaus Kähler Holst

References

Rosenblum & van der Laan (2010) Simple, Efficient Estimators of Treatment Effects in Randomized Trials Using Generalized Linear Models to Leverage Baseline Variables, The International Journal of Biostatistics

See Also

Trial

Examples

```
trial <- Trial$new(
    covariates = function(n) data.frame(a = rbinom(n, 1, 0.5), x = rnorm(n)),
    outcome = setargs(outcome_count,
      mean = ~ 1 + a*x,
      par = c(1, -0.1, 0.5, 0.2),
      overdispersion = 2)
)
dd <- trial$simulate(3e2)

# crude mean comparison between arms (default behavior; y ~ a)
est <- est_glm(family = poisson)
est(dd)

# linear adjustment with one covariate (y ~ a + x)
est <- est_glm(family = poisson, covariates = "x")
est(dd)

# return estimates of all linear coefficients (useful for debugging)
est <- est_glm(family = poisson, covariates = "x", target.parameter = NULL)
est(dd)

# comparing robust and non-robust standard errors of poisson estimator by
# passing robust argument via ... to lava::estimate
estimators <- list(
  robust = est_glm(family = poisson),
  non.robust = est_glm(family = poisson, robust = FALSE)
)
res <- do.call(rbind, lapply(estimators, \(est) est(dd)$coefmat))
rownames(res) <- names(estimators)
res

dd_factor <- dd
dd_factor$a <- as.factor(dd_factor$a)
# target parameter needs to be changed because the name of the estimated
# regression coefficient changes when encoding the treatment variable as a
# factor
est_glm(family = poisson, target.parameter = "a1")(dd_factor)
```

est_phreg — *Marginal Cox proportional hazards model for the treatment effect in RCT*

Description

Marginal Cox proportional hazards model for the treatment effect in RCT

Usage

```
est_phreg(
  response = "Surv(time, status)",
  treatment = "a",
  level = 0.95,
  id = NULL
)
```

Arguments

`response`	Response variable (character or formula). Default: "Surv(time, status)"
`treatment`	Treatment variable (character)
`level`	Confidence interval level
`id`	Optional subject id variable (character)

Value

function

Author(s)

Klaus Kähler Holst

See Also

Trial est_adj

`get_factor_levels` *Get levels for factor columns in data.table*

Description

Get levels for factor columns in data.table

Usage

```
get_factor_levels(data)
```

Arguments

`data`	Covariate `data.table`

`optim_sa` *Root solver by Stochastic Approximation*

Description

Root solver by Stochastic Approximation

Usage

```
optim_sa(
  f,
  init = 0,
  function.args = list(),
  ...,
  method = c("standard", "discrete", "interpolate"),
  projection = identity,
  control = list(niter = 100, alpha = 0.25, stepmult = 1),
  verbose = TRUE,
  burn.in = 0.75
)
```

Arguments

`f`	Stochastic function
`init`	Initial value to evaluate `f` in
`function.args`	Additional arguments passed to `f`
`...`	Additional arguments passed to `f`
`method`	Method ('standard': standard Robbins-Monro, 'discrete' or 'interpolate' for integer stochastic optimization. See details section)

`projection`	Optional projection function that can be used to constrain the parameter value (e.g., function(x) max(x, tau)). Applied at the end of each iteration of the optimization.
`control`	Control arguments (`niter` number of iterations, `alpha` and `stepmult` control the step-length of the Polyak-Ruppert algorithm as described in the details section).
`verbose`	if TRUE additional messages are printed throughout the optimization
`burn.in`	Burn-in (fraction of initial values to disregard) when applying Polyak–Ruppert averaging (alpha<1). Alternatively, `burn.in` can be given as an integer defining the absolut number of iterations to disregard.

Details

The aim is to find the root of the function $M(\theta) = Ef(\theta)$, where we are only able to observe the stochastic function $f(\theta)$. We will assume that $M$ is non-decreasing with a unique root θ^*. The Robbins-Monro algorithm works through the following update rule from an initial start value θ_0:

$$\theta_{n+1} = \theta_n - a_n f(\theta_n)$$

where $\sum_{n=0}^{\infty} a_n = \infty$ and $\sum_{n=0}^{\infty} a_n^2 < \infty$.

By averaging the iterates

$$\overline{\theta}_n = \frac{1}{n}\sum_{i=1}^{n-1} \theta_i$$

we can get improved stability of the algorithm that is less sensitive to the choice of the step-length a_n. This is known as the Polyak-Ruppert algorithm and to ensure convergence longer step sizes must be made which can be guaranteed by using $a_n = Kn^{-\alpha}$ with $0 < \alpha < 1$. The parameters K and α are controlled by the `stepmult` and `alpha` parameters of the `control` argument.

For discrete problems where θ must be an integer, we follow (Dupac & Herkenrath, 1984). Let $\lfloor x \rfloor$ denote the integer part of x, and define either (method="discrete"):

$$g(\theta) = I(U < \theta - \lfloor\theta\rfloor) f(\lfloor\theta\rfloor) + I(U \geq \theta - \lfloor\theta\rfloor) f(\lfloor\theta\rfloor + 1)$$

where $U \sim Uniform([0,1])$, or (method="interpolate"):

$$g(\theta) = (1 - \theta + \lfloor\theta\rfloor) f(\lfloor\theta\rfloor) + (\theta - \lfloor\theta\rfloor) f(\lfloor\theta\rfloor + 1).$$

The stochastic approximation method is then applied directly on g.

Dupač, V., & Herkenrath, U. (1984). On integer stochastic approximation. Aplikace matematiky, 29(5), 372-383.

`outcome_binary` *Simulate from binary model given covariates*

Description

Simulate from binary model with probability

$$\pi = g(\text{par}^\top X)$$

where X is the design matrix specified by the formula, and g is the link function specified by the family argument

Usage

```
outcome_binary(
  data,
  mean = NULL,
  par = NULL,
  outcome.name = "y",
  remove = c("id", "num"),
  family = binomial(logit),
  ...
)
```

Arguments

`data`	(data.table) Covariate data, usually the output of the covariate model of a Trial object.
`mean`	(formula, function) Either a formula specifying the design from 'data' or a function that maps `data` to the conditional mean value on the link scale (see examples). If NULL all main-effects of the covariates will be used, except columns that are defined via the `remove` argument.
`par`	(numeric) Regression coefficients (default zero). Can be given as a named list corresponding to the column names of `model.matrix`
`outcome.name`	Name of outcome variable ("y")
`remove`	Variables that will be removed from input `data` (if formula is not specified).
`family`	exponential family (default `binomial(logit)`)
`...`	Additional arguments passed to `mean` function (see examples)

Value

data.table

See Also

outcome_count outcome_continuous

Examples

```
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = outcome_binary
)
est <- function(data) glm(y ~ a, data = data, family = binomial(logit))
trial$simulate(1e4, mean = ~ 1 + a, par = c(1, 0.5)) |> est()

# default behavior is to set all regression coefficients to 0
trial$simulate(1e4, mean = ~ 1 + a) |> est()

# intercept defaults to 0 and regression coef for a takes the provided value
trial$simulate(1e4, mean = ~ 1 + a, par = c(a = 0.5)) |> est()
# trial$simulate(1e4, mean = ~ 1 + a, par = c("(Intercept)" = 1))

# define mean model that directly works on whole covariate data, incl id and
# num columns
trial$simulate(1e4, mean = \(x) with(x, lava::expit(1 + 0.5 * a))) |>
  est()

# par argument of outcome_binary is not passed on to mean function
trial$simulate(1e4,
  mean = \(x,  reg.par) with(x, lava::expit(reg.par[1] + reg.par[2] * a)),
  reg.par = c(1, 0.8)
) |> est()
```

`outcome_continuous` *Simulate from continuous outcome model given covariates*

Description

Simulate from continuous outcome model with mean

$$g(\text{par}^\top X)$$

where X is the design matrix specified by the formula, and g is the link function specified by the family argument

Usage

```
outcome_continuous(
  data,
  mean = NULL,
  par = NULL,
  sd = 1,
  het = 0,
  outcome.name = "y",
  remove = c("id", "num"),
  family = gaussian(),
  ...
)
```

Arguments

`data`	(data.table) Covariate data, usually the output of the covariate model of a Trial object.
`mean`	(formula, function) Either a formula specifying the design from 'data' or a function that maps `data` to the conditional mean value on the link scale (see examples). If NULL all main-effects of the covariates will be used, except columns that are defined via the `remove` argument.
`par`	(numeric) Regression coefficients (default zero). Can be given as a named list corresponding to the column names of `model.matrix`
`sd`	(numeric) standard deviation of Gaussian measurement error
`het`	Introduce variance hetereogeneity by adding a residual term $het \cdot \mu_x \cdot e$, where μ_x is the mean given covariates and e is an independent standard normal distributed variable. This term is in addition to the measurement error introduced by the `sd` argument.
`outcome.name`	Name of outcome variable ("y")
`remove`	Variables that will be removed from input `data` (if formula is not specified).
`family`	exponential family (default `gaussian(identity)`)
`...`	Additional arguments passed to `mean` function (see examples)

Value

data.table

See Also

outcome_count outcome_binary

Examples

```
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), x = rnorm(n)),
  outcome = outcome_continuous
)
est <- function(data) glm(y ~ a + x, data = data)
trial$simulate(1e4, mean = ~ 1 + a + x, par = c(1, 0.5, 2)) |> est()

# default behavior is to set all regression coefficients to 0
trial$simulate(1e4, mean = ~ 1 + a + x) |> est()

# intercept defaults to 0 and regression coef for a takes the provided value
trial$simulate(1e4, mean = ~ 1 + a, par = c(a = 0.5)) |> est()
# trial$simulate(1e4, mean = ~ 1 + a, par = c("(Intercept)" = 0.5)) |> est()

# define mean model that directly works on whole covariate data, incl id and
# num columns
trial$simulate(1e4, mean = \(x) with(x, -1 + a * 2 + x * -3)) |>
  est()
```

```
# par argument is not passed on to mean function
trial$simulate(1e4,
  mean = \(x,  reg.par) with(x, reg.par[1] + reg.par[2] * a),
  reg.par = c(1, 5)
) |> est()
```

`outcome_count` *Simulate from count model given covariates*

Description

Simulate from count model with intensity

$$\lambda = \text{exposure-time}\exp(\text{par}^\top X)$$

where X is the design matrix specified by the formula

Usage

```
outcome_count(
  data,
  mean = NULL,
  par = NULL,
  outcome.name = "y",
  exposure = 1,
  remove = c("id", "num"),
  zero.inflation = NULL,
  overdispersion = NULL,
  ...
)
```

Arguments

`data`	(data.table) Covariate data, usually the output of the covariate model of a Trial object.
`mean`	(formula, function) Either a formula specifying the design from 'data' or a function that maps `data` to the conditional mean value on the link scale (see examples). If NULL all main-effects of the covariates will be used, except columns that are defined via the `remove` argument.
`par`	(numeric) Regression coefficients (default zero). Can be given as a named list corresponding to the column names of `model.matrix`
`outcome.name`	Name of outcome variable ("y")
`exposure`	Exposure times. Either a scalar, vector or function.
`remove`	Variables that will be removed from input `data` (if formula is not specified).
`zero.inflation`	vector of probabilities or a function of the covariates 'x' including an extra column 'rate' with the rate parameter.
`overdispersion`	variance of gamma-frailty either given as a numeric vector or a function of the covariates 'x' with an extra column 'rate' holding the rate parameter 'rate'
`...`	Additional arguments passed to `mean` and `exposure` function

Value

data.table

See Also

outcome_binary outcome_continuous

Examples

```
covariates <- function(n) data.frame(a = rbinom(n, 1, 0.5), x = rnorm(n))
trial <- Trial$new(covariates = covariates, outcome = outcome_count)
trial$args_model(
  mean = ~ 1 + a + x,
  par = c(2.5, 0.65, 0),
  overdispersion = 1 / 2,
  exposure = 2 # identical exposure time for all subjects
)
est <- function(data) {
  glm(y ~ a + x + offset(log(exposure)), data, family = poisson())
}
trial$simulate(1e4) |> est()

# intercept + coef for x default to 0 and regression coef for a takes
# the provided value
trial$simulate(1e4, par = c(a = 0.65)) |> est()
# trial$simulate(1e4, mean = ~ 1 + a, par = c("(Intercept)" = 1))

# define mean model that directly works on whole covariate data, incl id and
# num columns
trial$simulate(1e4, mean = \(x) with(x, exp(1 + 0.5 * a))) |>
  est()

# treatment-dependent exposure times
trial$simulate(1e4, exposure = function(dd) 1 - 0.5 * dd$a) |>
  head()
```

`outcome_lp` *Calculate linear predictor from covariates*

Description

Calculate linear predictor

$$\text{par}^\top X$$

where X is the design matrix specified by the formula

Usage

```
outcome_lp(
  data,
  mean = NULL,
  par = NULL,
  model = NULL,
  offset = NULL,
  treatment = NULL,
  intercept = TRUE,
  default.parameter = 0,
  family = gaussian(),
  remove = c("id", "num"),
  ...
)
```

Arguments

`data`	(data.table) Covariate data, usually the output of the covariate model of a Trial object.
`mean`	formula specifying design from 'data' or a function that maps x to the mean value. If NULL all main-effects of the covariates will be used
`par`	(numeric) Regression coefficients (default zero). Can be given as a named list corresponding to the column names of `model.matrix`
`model`	Optional model object (glm, mets::phreg, ...)
`offset`	Optional offset variable name
`treatment`	Optional name of treatment variable
`intercept`	When FALSE the intercept will removed from the design matrix
`default.parameter`	when `model` and `treatment` is specified, interaction terms between `treatment` and all other covariates in `model` is added to the simulation model. `default.parameter` specifies the default parameter of these extra parameters which can be changed individually with the `par` argument.
`family`	family (default 'gaussian(identity)'). The inverse link-function is used to map the mean to the linear predictor scale (if mean is given as a function)
`remove`	variables that will be removed from input data (if formula is not specified)
`...`	Additional arguments passed to `mean` function (see examples)

Value

data.table

outcome_phreg — *Outcome model for time-to-event end-points (proportional hazards)*

Description

Outcome model for time-to-event end-points (proportional hazards)

Usage

```
outcome_phreg(
  data,
  lp = NULL,
  par = NULL,
  outcome.name = c("time", "status"),
  remove = c("id", "num"),
  model = NULL,
  cens.model = NULL,
  cens.lp = NULL,
  cens.par = NULL,
  ...
)
```

Arguments

`data`	(data.table) Covariate data, usually the output of the covariate model of a Trial object.
`lp`	linear predictor (formula or function)
`par`	optional list of model parameter
`outcome.name`	names of outcome (time and censoring status)
`remove`	Variables that will be removed from input `data` (if formula is not specified).
`model`	optional mets::phreg object
`cens.model`	optional model for censoring mechanism
`cens.lp`	censoring linear predictor argument (formula or function)
`cens.par`	list of censoring model parameters
`...`	Additional arguments to outcome_lp

Value

data.table

Author(s)

Klaus Kähler Holst

outcome_recurrent *EXPERIMENTAL: Outcome model for recurrent events with terminal events end-points*

Description

This function is still in an experimental state where the interface and functionality might change in the future

Usage

```
outcome_recurrent(
  data,
  lp = NULL,
  par = NULL,
  outcome.name = c("time", "status"),
  remove = c("id", "num"),
  model = NULL,
  death.model = NULL,
  death.lp = NULL,
  death.par = NULL,
  cens.model = NULL,
  cens.lp = NULL,
  cens.par = NULL,
  ...
)
```

Arguments

`data`	data.frame (covariates)
`lp`	linear predictor (formula or function)
`par`	optional list of model parameter
`outcome.name`	names of outcome (time and censoring status)
`remove`	variables that will be removed from input data (if formula is not specified)
`model`	optional mets::phreg object
`death.model`	optional model for death (terminal) events
`death.lp`	optional death linear predictor argument (formula or function)
`death.par`	optional list of death model parameters
`cens.model`	optional model for censoring mechanism
`cens.lp`	optional censoring linear predictor argument (formula or function)
`cens.par`	optional list of censoring model parameters
`...`	Additional arguments to outcome_lp

Value

function (random generator)

`outcome_shared` *Outcome model*

Description

Outcome model

Arguments

`data`	(data.table) Covariate data, usually the output of the covariate model of a Trial object.
`par`	(numeric) Regression coefficients (default zero). Can be given as a named list corresponding to the column names of `model.matrix`
`outcome.name`	Name of outcome variable ("y")
`remove`	Variables that will be removed from input `data` (if formula is not specified).
`mean`	(formula, function) Either a formula specifying the design from 'data' or a function that maps `data` to the conditional mean value on the link scale (see examples). If NULL all main-effects of the covariates will be used, except columns that are defined via the `remove` argument.
`...`	Additional arguments passed to `mean` function (see examples)

Value

data.table

`rmvn` *Multivariate normal distribution function*

Description

Draw random samples from multivariate normal distribution with variance given by a correlation matrix.

Usage

```
rmvn(n, mean, cor, var = NULL)
```

Arguments

`n`	number of samples
`mean`	matrix with mean values (either a 1xp or nxp matrix)
`cor`	matrix with correlation (either a 1x((p-1)*p/2) or nx((p-1)*p/2) matrix. The correlation coefficients must be given in the order R(1,2), R(1,3), ..., R(1,p), R(2,3), ... R(2,p), ... where R(i,j) is the entry in row i and column j of the correlation matrix.
`var`	Optional covariance matrix (instead of 'cor' argument)

Examples

```
rmvn(10, cor = rep(c(-0.999, 0.999), each = 5))
```

`rnb` *Simulate from a negative binomial distribution*

Description

Parametrized by mean (rate) and variance. Both parameters can be vector arguments. For this case with mean = variance = c(r1, r2) and n = 5, the returned vector contains 5 Poisson samples. Three samples are drawn from a Poisson distribution with rate r1 (index 1, 3 and 5 in output vector) and two from a Poisson with rate r2 (index 2 and 4).

Usage

```
rnb(n, mean, variance = mean, gamma.variance = NULL)
```

Arguments

`n`	Number of samples (integer)
`mean`	Mean vector (rate parameter)
`variance`	Variance vector
`gamma.variance`	(optional) poisson-gamma mixture parametrization. Variance (vector) of gamma distribution with mean 1.

Value

Vector of n realizations

Examples

```
with(
  data.frame(x = rnb(1e4, mean = 100, var = 500)),
  c(mean = mean(x), var = var(x))
)
```

`sample_covar_parametric_model`
Sample from an estimated parametric covariate model

Description

Sample from an estimated parametric covariate model

Usage

```
sample_covar_parametric_model(n, model = NULL, model.path = NULL)
```

Arguments

`n`	Sample size
`model`	lava::lvm object with estimated coefficients
`model.path`	Path to dumped model object (RDS file) on disk (optional)

Value

data.table

Author(s)

Benedikt Sommer

Examples

```
data <- data.table::data.table(
  x = rnorm(1e3), y = as.factor(rbinom(1e3, size = 1, prob=0.5))
)

m <- estimate_covar_model_full_cond(data)
samples <- sample_covar_parametric_model(n=10, model = m)
print(head(samples))
```

`setargs` *Set default arguments of a function*

Description

Sets default values for arguments in `f`. Care should be taken when `missing` is used in `f` (see examples).

Usage

```
setargs(f, ..., setargs.warn = TRUE)
```

Arguments

`f`	function
`...`	arguments to set
`setargs.warn`	cast warning when trying to set default values for arguments that do not exist in `f`

Author(s)

Benedikt Sommer

Examples

```
foo <- function(x, a = 5, ...) {
  foo1 <- function(x, b = 5) return(b)
  c(a = a, b = foo1(x, ...), x = x)
}
foo(1)

f <- setargs(foo, a = 10) # set new default value for a
f(1)

# default value of b in lower-level function is unaffected and warning is
# cast to inform that b is not an argument in f
f <- setargs(foo, b = 10)
f(1)
# disable warning message
setargs(foo, b = 10, setargs.warn = FALSE)(1)

# arguments of lower-level functions can be set with setallargs
f <- setallargs(foo, a = 10, b = 10)
f(1)

# does not work when `missing` checks for missing formal arguments.
foo1 <- function(x, a) {
  if (missing(a)) a <- 5
  return(c(x = x, a = a))
}
f <- setargs(foo1, a = 10)
f(1)
```

`Trial` *R6 class for power and sample-size calculations for a clinical trial*

Description

Simulation of RCT with flexible covariates distributions simulation.

Public fields

`info` Optional information/name of the model

`covariates` covariate generator (function of sample-size n and optional parameters)

`outcome_model` Generator for outcome given covariates (function of covariates x in long format)

`exclusion` function defining exclusion criterion

`estimates` (trial.estimates) Parameter estimates of Monte-Carlo simulations returned by Trial$run(). The field is flushed, i.e. set to its default value *NULL*, when model arguments (Trial$args_model()) or estimators (Trial$estimators()) are modified.

Methods

Public methods:

- `Trial$new()`
- `Trial$args_model()`
- `Trial$args_summary()`
- `Trial$estimators()`
- `Trial$simulate()`
- `Trial$run()`
- `Trial$estimate_power()`
- `Trial$estimate_samplesize()`
- `Trial$summary()`
- `Trial$print()`
- `Trial$clone()`

Method `new()`**:** Initialize new Trial object

Usage:

```
Trial$new(
  outcome,
  covariates = NULL,
  exclusion = identity,
  estimators = list(),
  summary.args = list(),
  info = NULL
)
```

Arguments:

`outcome` outcome model given covariates (the first positional argument must be the covariate data)

`covariates` covariate simulation function (must have 'n' as first named argument and returns either a list of data.table (data.frame) for each observation period or a single data.table (data.frame) in case of a single observation period)

`exclusion` function describing selection criterion (the first positional argument must be the combined covariate and outcome data and the function must return the subjects who are included in the trial)

`estimators` optional list of estimators or single estimator function `summary.args` list of arguments that override default values in Trial$summary() when power and sample sizes are estimated with Trial$estimate_power() and Trial$estimate_samplesize()

`info` optional string describing the model

Method `args_model()`**:** Get, specify or update parameters of covariate, outcome and exclusion model. Parameters are set in a named list, and updated when parameter names match with existing values in the list.

Usage:

```
Trial$args_model(.args = NULL, .reset = FALSE, ...)
```

Arguments:

`.args` (list or character) named list of arguments to update or set. A single or subset of arguments can be retrieved by passing the respective argument names as a character or character vector.

`.reset` (logical or character) Reset all or a subset of previously set parameters. Can be combined with setting new parameters.

`...` Alternative to using `.args` to update or set arguments

Examples:

```
trial <- Trial$new(
  covariates = function(n, p = 0.5) data.frame(a = rbinom(n, 1, p)),
  outcome = function(data, ate, mu) rnorm(nrow(data), mu + data$a * ate)
)

# set and update parameters
trial$args_model(.args = list(ate = 2, p = 0.5, mu = 3))
trial$args_model(ate = 5, p = 0.6) # update parameters
trial$args_model(list(ate = 2), p = 0.5) # combine first arg with ...

# retrieve parameters
trial$args_model() # return all set parameters
trial$args_model("ate") # select a single parameter
trial$args_model(c("ate", "mu")) # multiple parameters

# remove parameters
trial$args_model(.reset = "ate") # remove a single parameter
trial$args_model(.reset = TRUE) # remove all parameters

# remove and set/update parameters
trial$args_model(ate = 2, p = 0.5, .reset = TRUE)
trial$args_model(ate = 5, .reset = "p") # removing p and updating ate
```

Method `args_summary()`**:** Get, specify or update the summary.args attribute.

Usage:

```
Trial$args_summary(.args = NULL, .reset = FALSE, ...)
```

Arguments:

`.args` (list or character) named list of arguments to update or set. A single or subset of arguments can be retrieved by passing the respective argument names as a character or character vector.

`.reset` (logical or character) Reset all or a subset of previously set parameters. Can be combined with setting new parameters.

`...` Alternative to using `.args` to update or set arguments

Examples:

```
trial <- Trial$new(
  covariates = function(n, p = 0.5) data.frame(a = rbinom(n, 1, p)),
  outcome = function(data, ate, mu) rnorm(nrow(data), mu + data$a * ate)
)
# set and update parameters
trial$args_summary(list(level = 0.05, alternative = "<"))
trial$args_summary(level = 0.25) # update parameters

# retrieve parameters
trial$args_summary() # return all set parameters
trial$args_summary("level") # select a single parameter
trial$args_summary(c("level", "alternative")) # multiple parameters

# remove parameters
trial$args_summary(.reset = "level") # remove a single parameter
trial$args_summary(.reset = TRUE) # remove all parameters

# remove and set/update parameters
trial$args_summary(alternative = "!=", level = 0.05, .reset = TRUE)
# removing alternative and setting level
trial$args_summary(level = 0.05, .reset = "alternative")
```

Method `estimators()`**:** Get, specify or update estimators.

Usage:

```
Trial$estimators(.estimators = NULL, .reset = FALSE, ...)
```

Arguments:

`.estimators` (list, function or character) Argument controlling the getter or setter behavior. Estimators are set or updated by providing a single estimator (function) or list of estimators, and retrieved by providing a character (see examples).

`.reset` (logical or character) Reset all or a subset of previously set estimators. Can be combined with setting new estimators.

`...` Alternative to `.estimators` for updating or setting estimators.

Details: A name is internally assigned to estimators when calling the method with `.estimators` set to a single estimator or a list with unnamed elements. The names are a combination of an *est* prefix and an integer that indicates the *n*th added unnamed estimator.

Examples:

```
estimators <- list(marginal = est_glm(), adj = est_glm(covariates = "x"))
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), x = rnorm(n)),
  outcome = \(data, ate = -0.5) rnorm(nrow(data), data$a * ate),
  estimators = estimators
)
```

```
# get estimators
trial$estimators() |> names() # list all estimators
trial$estimators("marginal") |> names() # select a single estimator
trial$estimators(c("marginal", "adj")) |> names() # select mult. est.

# remove estimators
trial$estimators(.reset = "marginal") # remove a single estimator
trial$estimators(.reset = TRUE) # remove all estimators

# set or update estimators
trial$estimators(estimators)
trial$estimators(adj2 = est_adj(covariates = "x")) # add new estimator
# update adj and remove adj2
trial$estimators(adj = est_glm(covariates = "x"), .reset = "adj2")

# unnamed estimators (adding default name)
estimators <- list(est_glm(), est_glm(covariates = "x"))
trial$estimators(estimators, .reset = TRUE)
trial$estimators(.reset = "est1")
trial$estimators(est_glm()) # replaces removed est1
```

Method `simulate()`**:** Simulate data by applying parameters to the trial model. The method samples first from the covariate model. Outcome data sampling follows by passing the simulated covariate data to the outcome model. An optional exclusion model is applied to the combined covariates and outcomes data. The sampling process is repeated in case any subjects are removed by the exclusion model until a total of `n` subjects are sampled or the maximum iteration number `.niter` is reached.

The method adds two auxiliary columns to the simulated data to identify distinct patients (*id*) and periods (*num*) in case of time-dependent covariate and outcome models. The columns are added to the sampled covariate data before sampling the outcomes. A *data.table* with both columns is provided to the outcome model in case no covariate model is defined. Thus, the outcome model is always applied to at least a *data.table* with an *id* and *num* column. The default column name *y* is used for the outcome variable in the returned *data.table* when the defined outcome model returns a vector. The name is easily changed by returning a *data.table* with a named column (see examples).

The optional argument `...` of this method can be used to provide parameters to the trial model as an addition to parameters that have already been defined via Trial$args_model(). Data is simulated from the union of parameters, where parameters provided via the optional argument of this method take precedence over parameters defined via Trial$args_model(). However, parameters that have been set via Trial$args_model() are not updated when optional arguments are provided.

Usage:

```
Trial$simulate(n, .niter = 500L, ...)
```

Arguments:

`n` (integer) Number of observations (sample size)

`.niter` (integer) Maximum number of simulation runs to avoid infinite loops for ill defined exclusion functions.

`...` Arguments to covariate, outcome and exclusion model functions

Returns: data.table with `n` rows

Examples:

```
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate))
)

# applying and modifying parameters
n <- 10
trial$simulate(n) # use parameters set during initialization
trial$args_model(ate = -100) # update parameters
trial$simulate(n)
trial$simulate(n, ate = 100) # change ate via optional argument

# rename outcome variable
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) {
    data.frame(yy = with(data, rnorm(nrow(data), a * ate)))
  }
)
trial$simulate(n)

# return multiple outcome variables
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), y.base = rnorm(n)),
  outcome = \(data, ate = 0) {
    y  <-  with(data, rnorm(nrow(data), a * ate))
    return(data.frame(y = y, y.chg = data$y.base - y))
  }
)
trial$simulate(n)

# use exclusion argument to post-process sampled outcome variable to
# achieve the same as in the above example but without modifying the
# originally defined outcome model
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), y.base = rnorm(n)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate)),
  exclusion = \(data) {
   cbind(data, data.frame(y.chg = data$y.base - data$y))
  }
)
trial$simulate(n)

# no covariate model
trial <- Trial$new(
  outcome = \(data, ate = 0) {
    n <- nrow(data)
    a <- rbinom(n, 1, 0.5)
    return(data.frame(a = a, y = rnorm(n, a * ate)))
  }
)
trial$simulate(n)
```

Method `run()`**:** Run trial and estimate parameters multiple times

The method calls Trial$simulate() `R` times and applies the specified estimators to each simulated dataset of sample size `n`. Parameters to the covariates, outcome and exclusion models can be provided as optional arguments to this method call in addition to parameters that have already been defined via Trial$args_model(). The behavior is identical to Trial$simulate(), except that `.niter` can be provided as an optional argument to this method for controlling the maximum number of simulation runs in Trial$simulate().

Estimators fail silently in that errors occurring when applying an estimator to each simulated dataset will not terminate the method call. The returned trial.estimates object will instead indicate that estimators failed.

Usage:

```
Trial$run(n, R = 100, estimators = NULL, ...)
```

Arguments:

`n` (integer) Number of observations (sample size)

`R` (integer) Number of replications

`estimators` (list or function) List of estimators or a single unnamed estimator

`...` Arguments to covariate, outcome and exclusion model functions

Returns: (invisible) An object of class trial.estimates, which contains the estimates of all estimators and all information to repeat the simulation. The return object is also assigned to the `estimates` field of this Trial class object (see examples).

Examples:

```
\donttest{
# future::plan("multicore")
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate)),
  estimators = list(glm = est_glm())
)
trial$args_summary(alternative = "<")

# the returned trial.estimates object contains the estimates for each of
# the R simulated data sets + all necessary information to re-run the
# simulation
res <- trial$run(n = 100, R = 50) # store return object in a new variable
print(trial$estimates) # trial$estimates == res

# the basic usage is to apply the summary method to the generated
# trial.estimates object.
trial$summary()

# combining Trial$run and summary is faster than using
# Trial$estimate_power when modifying only the parameters of the
# decision-making function
sapply(
  c(0, 0.25, 0.5),
  \(ni) trial$summary(ni.margin = ni)[, "power"]
)

# changing the ate parameter value
trial$run(n = 100, R = 50, ate = -0.2)

# supplying another estimator
trial$run(n = 100, R = 50, estimators = est_glm(robust = FALSE))
}
```

Method `estimate_power()`**:** Estimates statistical power for a specified trial

Convenience method that first runs Trial$run() and subsequently applies Trial$summary() to derive the power for each estimator. The behavior of passing arguments to lower level functions is identical to Trial$run().

Usage:

```
Trial$estimate_power(n, R = 100, estimators = NULL, summary.args = list(), ...)
```

Arguments:

`n`	(integer) Number of observations (sample size)
`R`	(integer) Number of replications
`estimators`	(list or function) List of estimators or a single unnamed estimator
`summary.args`	(list) Arguments passed to summary method for decision-making
`...`	Arguments to covariate, outcome and exclusion model functions

Returns: numeric

Examples:

```
\donttest{
# toy examples with small number of Monte-Carlo replicates
# future::plan("multicore")
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate)),
  estimators = list(glm = est_glm())
)
trial$args_summary(alternative = "<")

# using previously defined estimators and summary.args
trial$estimate_power(n = 100, R = 20)

# supplying parameters to outcome function
trial$estimate_power(n = 100, R = 20, ate = -100)
# modifying arguments of summary function
trial$estimate_power(n = 100, R = 20, ate = -100,
 summary.args = list(alternative = ">")
)

# supplying estimators to overrule previously set estimators
trial$estimate_power(n = 100, R = 20,
 estimators = list(est_glm(), est_adj()))
}
```

Method `estimate_samplesize()`**:** Estimate the minimum sample-size required to reach a desired statistical power with a specified estimator. An initial rough estimate is obtained via bisection, followed by a stochastic approximation (Robbins-Monro) algorithm, and finally, a grid search (refinement step) in the neighborhood of the current best solution.

Note that the estimation procedure for the sample size will not populate the estimates attribute of a trial object.

Usage:

```
Trial$estimate_samplesize(
  ...,
  power = 0.9,
  estimator = NULL,
  interval = c(50L, 10000L),
  bisection.control = list(R = 100, niter = 6),
  sa.control = list(R = 1, niter = 250, stepmult = 100, alpha = 0.5),
  refinement = seq(-10, 10, by = 5),
  R = 1000,
  interpolate = TRUE,
  verbose = TRUE,
  minimum = 10L,
  summary.args = list()
)
```

Arguments:

`...` Arguments to covariate, outcome and exclusion model functions

`power` (numeric) Desired statistical power

`estimator` (list or function) Estimator (function) to be applied. If NULL, then estimate sample size for all estimators defined via Trial$estimators(). A prefix *est* is used to label unnamed estimators.

`interval` (integer vector) Interval in which to initially look for a solution with the bisection algorithm. Passing an integer will skip the bisection algorithm and use the provided integer as the initial solution for the stochastic approximation algorithm

`bisection.control` (list) Options controlling the bisection algorithm (bisection). Default values can also be changed for a subset of options only (see examples).

`sa.control` (list) Options controlling the stochastic approximation (Robbins-Monro) algorithm (optim_sa). Default values can also be changed for a subset of options only (see examples).

`refinement` (integer vector) Vector to create an interval whose center is the sample size estimate of the Robbins-Monro algorithm.

`R` (integer) Number of replications to use in Monte Carlo simulation of refinement calculations.

`interpolate` (logical) If TRUE, a linear interpolation of the refinement points will be used to estimate the power.

`verbose` (logical) If TRUE, additional output will be displayed.

`minimum` (integer) Minimum sample size.

`summary.args` (list) Arguments passed to summary method for decision-making

Returns: samplesize_estimate S3 object

Examples:

```
\donttest{
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate, sd) with(data, rnorm(nrow(data), a * ate, sd)),
  estimators = list(marginal = est_glm())
)
trial$args_model(ate = -1, sd = 5)
trial$args_summary(alternative = "<")

# supply model parameter and estimator to call to overwrite previously
# set values
trial$estimate_samplesize(ate = -2, estimator = est_glm())

# reduce number of iterations for bisection step but keep R = 100
# (default value)
# trial$estimate_samplesize(bisection.control = list(niter = 2))

# reduce significance level from 0.05 to 0.025, but keep alternative as
# before
# trial$estimate_samplesize(summary.args = list(level = 0.025))
}
```

Method `summary()`**:** Summarize Monte Carlo studies of different estimators for the treatment effect in a randomized clinical trial. The method reports the power of both superiority tests (one-sided or two-sided) and non-inferiority tests, together with summary statistics of the different estimators.

Usage:

```
Trial$summary(
  level = 0.05,
  null = 0,
  ni.margin = NULL,
  alternative = "!=",
  reject.function = NULL,
  true.value = NULL,
  nominal.coverage = 0.9,
  estimates = NULL,
  ...
)
```

Arguments:

`level` (numeric) significance level

`null` (numeric) null hypothesis to test

`ni.margin` (numeric) non-inferiority margin

`alternative` alternative hypothesis (not equal !=, less <, greater >)

`reject.function` Optional function calculating whether to reject the null hypothesis

`true.value` Optional true parameter value

`nominal.coverage` Width of confidence limits

`estimates` Optional trial.estimates object. When provided, these estimates will be used instead of the object's stored estimates. This allows calculating summaries for different trial results without modifying the object's state.

`...` additional arguments to lower level functions

Returns: matrix with results of each estimator stored in separate rows

Examples:

```
outcome <- function(data, p = c(0.5, 0.25)) {
  a <- rbinom(nrow(data), 1, 0.5)
  data.frame(a = a, y = rbinom(nrow(data), 1, p[1] * (1 - a) + p[2] * a)
  )
}
trial <- Trial$new(outcome, estimators = est_glm())
trial$run(n = 100, R = 100)
# two-sided test with 0.05 significance level (alpha = 0.05) (default
# values)
trial$summary(level = 0.05, alternative = "!=")
# on-sided test
trial$summary(level = 0.025, alternative = "<")
# non-inferiority test
trial$summary(level = 0.025, ni.margin = -0.5)

# provide simulation results to summary method via estimates argument
res <- trial$run(n = 100, R = 100, p = c(0.5, 0.5))
trial$summary(estimates = res)

# calculate empirical bias, rmse and coverage for true target parameter
trial$summary(estimates = res, true.value = 0)
```

Method `print()`: Print method for Trial objects

Usage:

```
Trial$print(..., verbose = FALSE)
```

Arguments:

`...` Additional arguments to lower level functions (not used).

`verbose` (logical) By default, only print the function arguments of the covariates, outcome and exclusion models. If *TRUE*, then also print the function body.

Examples:

```
trial <- Trial$new(
  covariates = function(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = function(data, sd = 1) rnorm(nrow(data), data$a * -1, sd),
  estimators = list(marginal = est_glm()),
  info = "Some trial info"
)
trial$args_model(sd = 2)
trial$args_summary(level = 0.025)

print(trial) # only function headers
print(trial, verbose = TRUE) # also print function bodies
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Trial$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

Klaus Kähler Holst, Benedikt Sommer

Benedikt Sommer

Klaus Kähler Holst

Examples

```
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), x = rnorm(n)),
  outcome = setargs(outcome_count, par = c(1, 0.5, 1), overdispersion = 0.7)
)

trial$estimators(
  unadjusted = est_glm(family = "poisson"),
  adjusted = est_glm(family = "poisson", covariates = "x")
)

trial$run(n = 200, R = 100)
trial$summary()


## ------------------------------------------------
## Method `Trial$args_model`
## ------------------------------------------------

trial <- Trial$new(
  covariates = function(n, p = 0.5) data.frame(a = rbinom(n, 1, p)),
  outcome = function(data, ate, mu) rnorm(nrow(data), mu + data$a * ate)
)
```

```
# set and update parameters
trial$args_model(.args = list(ate = 2, p = 0.5, mu = 3))
trial$args_model(ate = 5, p = 0.6) # update parameters
trial$args_model(list(ate = 2), p = 0.5) # combine first arg with ...

# retrieve parameters
trial$args_model() # return all set parameters
trial$args_model("ate") # select a single parameter
trial$args_model(c("ate", "mu")) # multiple parameters

# remove parameters
trial$args_model(.reset = "ate") # remove a single parameter
trial$args_model(.reset = TRUE) # remove all parameters

# remove and set/update parameters
trial$args_model(ate = 2, p = 0.5, .reset = TRUE)
trial$args_model(ate = 5, .reset = "p") # removing p and updating ate

## ------------------------------------------------
## Method `Trial$args_summary`
## ------------------------------------------------

trial <- Trial$new(
  covariates = function(n, p = 0.5) data.frame(a = rbinom(n, 1, p)),
  outcome = function(data, ate, mu) rnorm(nrow(data), mu + data$a * ate)
)
# set and update parameters
trial$args_summary(list(level = 0.05, alternative = "<"))
trial$args_summary(level = 0.25) # update parameters

# retrieve parameters
trial$args_summary() # return all set parameters
trial$args_summary("level") # select a single parameter
trial$args_summary(c("level", "alternative")) # multiple parameters

# remove parameters
trial$args_summary(.reset = "level") # remove a single parameter
trial$args_summary(.reset = TRUE) # remove all parameters

# remove and set/update parameters
trial$args_summary(alternative = "!=", level = 0.05, .reset = TRUE)
# removing alternative and setting level
trial$args_summary(level = 0.05, .reset = "alternative")

## ------------------------------------------------
## Method `Trial$estimators`
## ------------------------------------------------

estimators <- list(marginal = est_glm(), adj = est_glm(covariates = "x"))
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), x = rnorm(n)),
  outcome = \(data, ate = -0.5) rnorm(nrow(data), data$a * ate),
  estimators = estimators
```

```
)

# get estimators
trial$estimators() |> names() # list all estimators
trial$estimators("marginal") |> names() # select a single estimator
trial$estimators(c("marginal", "adj")) |> names() # select mult. est.

# remove estimators
trial$estimators(.reset = "marginal") # remove a single estimator
trial$estimators(.reset = TRUE) # remove all estimators

# set or update estimators
trial$estimators(estimators)
trial$estimators(adj2 = est_adj(covariates = "x")) # add new estimator
# update adj and remove adj2
trial$estimators(adj = est_glm(covariates = "x"), .reset = "adj2")

# unnamed estimators (adding default name)
estimators <- list(est_glm(), est_glm(covariates = "x"))
trial$estimators(estimators, .reset = TRUE)
trial$estimators(.reset = "est1")
trial$estimators(est_glm()) # replaces removed est1

## ------------------------------------------------
## Method `Trial$simulate`
## ------------------------------------------------

trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate))
)

# applying and modifying parameters
n <- 10
trial$simulate(n) # use parameters set during initialization
trial$args_model(ate = -100) # update parameters
trial$simulate(n)
trial$simulate(n, ate = 100) # change ate via optional argument

# rename outcome variable
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) {
    data.frame(yy = with(data, rnorm(nrow(data), a * ate)))
  }
)
trial$simulate(n)

# return multiple outcome variables
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), y.base = rnorm(n)),
  outcome = \(data, ate = 0) {
    y  <-  with(data, rnorm(nrow(data), a * ate))
    return(data.frame(y = y, y.chg = data$y.base - y))
  }
)
trial$simulate(n)

# use exclusion argument to post-process sampled outcome variable to
# achieve the same as in the above example but without modifying the
# originally defined outcome model
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5), y.base = rnorm(n)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate)),
  exclusion = \(data) {
   cbind(data, data.frame(y.chg = data$y.base - data$y))
  }
)
trial$simulate(n)

# no covariate model
trial <- Trial$new(
  outcome = \(data, ate = 0) {
    n <- nrow(data)
    a <- rbinom(n, 1, 0.5)
    return(data.frame(a = a, y = rnorm(n, a * ate)))
  }
)
trial$simulate(n)

## ------------------------------------------------
## Method `Trial$run`
## ------------------------------------------------


# future::plan("multicore")
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate)),
  estimators = list(glm = est_glm())
)
trial$args_summary(alternative = "<")

# the returned trial.estimates object contains the estimates for each of
# the R simulated data sets + all necessary information to re-run the
# simulation
res <- trial$run(n = 100, R = 50) # store return object in a new variable
print(trial$estimates) # trial$estimates == res

# the basic usage is to apply the summary method to the generated
# trial.estimates object.
trial$summary()

# combining Trial$run and summary is faster than using
# Trial$estimate_power when modifying only the parameters of the
# decision-making function
```

```
sapply(
  c(0, 0.25, 0.5),
  \(ni) trial$summary(ni.margin = ni)[, "power"]
)

# changing the ate parameter value
trial$run(n = 100, R = 50, ate = -0.2)

# supplying another estimator
trial$run(n = 100, R = 50, estimators = est_glm(robust = FALSE))


## ------------------------------------------------
## Method `Trial$estimate_power`
## ------------------------------------------------


# toy examples with small number of Monte-Carlo replicates
# future::plan("multicore")
trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate = 0) with(data, rnorm(nrow(data), a * ate)),
  estimators = list(glm = est_glm())
)
trial$args_summary(alternative = "<")

# using previously defined estimators and summary.args
trial$estimate_power(n = 100, R = 20)

# supplying parameters to outcome function
trial$estimate_power(n = 100, R = 20, ate = -100)

# modifying arguments of summary function
trial$estimate_power(n = 100, R = 20, ate = -100,
 summary.args = list(alternative = ">")
)

# supplying estimators to overrule previously set estimators
trial$estimate_power(n = 100, R = 20,
 estimators = list(est_glm(), est_adj()))


## ------------------------------------------------
## Method `Trial$estimate_samplesize`
## ------------------------------------------------


trial <- Trial$new(
  covariates = \(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = \(data, ate, sd) with(data, rnorm(nrow(data), a * ate, sd)),
  estimators = list(marginal = est_glm())
)
trial$args_model(ate = -1, sd = 5)
```

```
trial$args_summary(alternative = "<")

# supply model parameter and estimator to call to overwrite previously
# set values
trial$estimate_samplesize(ate = -2, estimator = est_glm())

# reduce number of iterations for bisection step but keep R = 100
# (default value)
# trial$estimate_samplesize(bisection.control = list(niter = 2))

# reduce significance level from 0.05 to 0.025, but keep alternative as
# before
# trial$estimate_samplesize(summary.args = list(level = 0.025))


## ------------------------------------------------
## Method `Trial$summary`
## ------------------------------------------------

outcome <- function(data, p = c(0.5, 0.25)) {
  a <- rbinom(nrow(data), 1, 0.5)
  data.frame(a = a, y = rbinom(nrow(data), 1, p[1] * (1 - a) + p[2] * a)
  )
}
trial <- Trial$new(outcome, estimators = est_glm())
trial$run(n = 100, R = 100)
# two-sided test with 0.05 significance level (alpha = 0.05) (default
# values)
trial$summary(level = 0.05, alternative = "!=")
# on-sided test
trial$summary(level = 0.025, alternative = "<")
# non-inferiority test
trial$summary(level = 0.025, ni.margin = -0.5)

# provide simulation results to summary method via estimates argument
res <- trial$run(n = 100, R = 100, p = c(0.5, 0.5))
trial$summary(estimates = res)

# calculate empirical bias, rmse and coverage for true target parameter
trial$summary(estimates = res, true.value = 0)

## ------------------------------------------------
## Method `Trial$print`
## ------------------------------------------------

trial <- Trial$new(
  covariates = function(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = function(data, sd = 1) rnorm(nrow(data), data$a * -1, sd),
  estimators = list(marginal = est_glm()),
  info = "Some trial info"
)
trial$args_model(sd = 2)
trial$args_summary(level = 0.025)
```

```
print(trial) # only function headers
print(trial, verbose = TRUE) # also print function bodies
```

`trial.estimates-class` *trial.estimates class object*

Description

Trial$run() returns an S3 class object `trial.estimates`. The object contains all information to reproduce the estimates as shown in the example. The object is a list with the following components:

model Trial object used to generate the estimates.

estimates (list) Estimates of Monte-Carlo runs for each of the estimators.

sample.data (data.table) Sample data returned from Trial$simulate().

estimators (list) Estimators applied to simulated data in each Monte-Carlo run.

sim.args (list) Arguments passed on to Trial$simulate() when simulating data in each Monte-Carlo run.

R (numeric) Number of Monte-Carlo replications.

S3 generics

The following S3 generic functions are available for an object of class `trial.estimates`:

`print` Basic print method.

Examples

```
trial <- Trial$new(
  covariates = function(n) data.frame(a = rbinom(n, 1, 0.5)),
  outcome = function(data) rnorm(nrow(data), data$a * -1)
 )
res <- trial$run(n = 100, R = 10, estimators = est_glm())
print(res)

# assuming previous estimates have been saved to disk.
# load estimates object and repeat simulation with more Monte-Carlo runs
res2 <- do.call(
  res$model$run,
  c(list(R = 20, estimators = res$estimators), res$sim.args)
)
print(res2)
```

Index