Package ‘adoptr’

June 7, 2024

Type Package

Title Adaptive Optimal Two-Stage Designs

Version 1.0.1

Description Optimize one or two-arm, two-stage designs for clinical trials with respect to several implemented objective criteria or custom objectives. Optimization under uncertainty and conditional (given stage-one outcome) constraints are supported.
See Pilz et al. (2019) <doi:10.1002/sim.8291> and
Kunzmann et al. (2021) <doi:10.18637/jss.v098.i09> for details.

License MIT + file LICENSE

Encoding UTF-8

Suggests knitr, rmarkdown, testthat, covr, rpact, vdiffr, pwr, dplyr, ggplot2, tidyr, gridExtra, bookdown

Imports nloptr, methods, glue

VignetteBuilder knitr

Collate adoptr.R util.R DataDistribution.R BinomialDistribution.R NormalDistribution.R StudentDistribution.R Prior.R PointMassPrior.R ContinuousPrior.R TwoStageDesign.R GroupSequentialDesign.R OneStageDesign.R Scores.R constraints.R minimize.R ConditionalPower.R ConditionalSampleSize.R regularization.R CompositeScore.R MaximumSampleSize.R

RoxygenNote 7.3.1

BugReports `https://github.com/optad/adoptr/issues`

URL `https://github.com/optad/adoptr`, `https://optad.github.io/adoptr/`

NeedsCompilation no

Author Kevin Kunzmann [aut, cph] (<`https://orcid.org/0000-0002-1140-7143`>),
Maximilian Pilz [aut, cre] (<`https://orcid.org/0000-0002-9685-1613`>),
Jan Meis [aut] (<`https://orcid.org/0000-0001-5407-7220`>),
Nico Bruder [aut]

Maintainer Maximilian Pilz <`maximilian.pilz@itwm.fraunhofer.de`>

Repository CRAN

Date/Publication 2024-06-07 14:40:11 UTC

Contents

adoptr *Adaptive Optimal Two-Stage Designs*

Description

The **adoptr** package provides functionality to explore custom optimal two-stage designs for one- or two-arm superiority tests. For more details on the theoretical background see doi:10.1002/sim.8291 and doi:10.18637/jss.v098.i09. **adoptr** makes heavy use of the S4 class system. A good place to start learning about it can be found here.

Quickstart

For a sample workflow and a quick demo of the capabilities, see here.

A more detailed description of the background and the usage of **adoptr** can be found here or here doi:10.18637/jss.v098.i09 .

A variety of examples is presented in the validation report hosted here.

Designs

adoptr currently supports TwoStageDesign, GroupSequentialDesign, and OneStageDesign.

Data distributions

Currently, the only implemented data distribution is Normal (one or two arms).

Priors

Both ContinuousPrior and PointMassPrior are supported for the single parameter of a DataDistribution.

Scores

See Scores for information on the basic system of representing scores. Available scores are ConditionalPower, ConditionalSampleSize, Power, and ExpectedSampleSize.

Author(s)

Maintainer: Maximilian Pilz <maximilian.pilz@itwm.fraunhofer.de> (ORCID)

Authors:

- Kevin Kunzmann <kevin.kunzmann@boehringer-ingelheim.com> (ORCID) [copyright holder]
- Jan Meis <meis@imbi.uni-heidelberg.de> (ORCID)
- Nico Bruder <bruder@imbi.uni-heidelberg.de>

See Also

Useful links:

- https://github.com/optad/adoptr
- https://optad.github.io/adoptr/
- Report bugs at https://github.com/optad/adoptr/issues

`AverageN2-class` *Regularization via L1 norm*

Description

Implements the L1-norm of the design's stage-two sample size function. The average of the stage-two sample size without weighting with the data distribution is computed. This can be interpreted as integration over a unifrom prior on the continuation region.

Usage

```
AverageN2(label = NA_character_)

## S4 method for signature 'AverageN2,TwoStageDesign'
evaluate(s, design, optimization = FALSE, subdivisions = 10000L, ...)
```

Arguments

`label`	object label (string)
`s`	`Score` object
`design`	object
`optimization`	logical, if `TRUE` uses a relaxation to real parameters of the underlying design; used for smooth optimization.
`subdivisions`	number of subdivisions to use for adaptive integration (only affects non-optimization code)
`...`	further optional arguments

Value

an object of class `AverageN2`

See Also

`N1` for penalizing n1 values

Examples

```
avn2 <- AverageN2()

evaluate(
   AverageN2(),
   TwoStageDesign(100, 0.5, 1.5, 60.0, 1.96, order = 5L)
) # 60
```

`Binomial-class` *Binomial data distribution*

Description

Implements the normal approximation for a test on rates. The reponse rate in the control group, r_C, has to be specified by `rate_control`. The null hypothesis is: $r_E <= r_C$, where r_E denotes the response rate in the invervention group. It is tested against the alternative $r_E > r_C$. The test statistic is given as $X_1 = \sqrt{n}(r_E - r_C)/\sqrt{2r_0(1 - r_0)}$, where r_0 denotes the mean between r_E and r_C in the two-armed case, and r_E in the one-armed case.#' All priors have to be defined for the rate difference $r_E - r_C$.

Usage

```
Binomial(rate_control, two_armed = TRUE)

## S4 method for signature 'Binomial'
quantile(x, probs, n, theta, ...)

## S4 method for signature 'Binomial,numeric'
simulate(object, nsim, n, theta, seed = NULL, ...)
```

Arguments

`rate_control`	assumed response rate in control group
`two_armed`	logical indicating if a two-armed trial is regarded
`x`	outcome
`probs`	vector of probabilities
`n`	sample size
`theta`	distribution parameter
`...`	further optional arguments
`object`	object of class `Binomial`
`nsim`	number of simulation runs
`seed`	random seed

Details

Note that `simulate` for class `Binomial` simulates the normal approximation of the test statistic.

Slots

`rate_control` cf. parameter 'rate_control'

See Also

see `probability_density_function` and `cumulative_distribution_function` to evaluate the pdf and the cdf, respectively.

Examples

```
datadist <- Binomial(rate_control = 0.2, two_armed = FALSE)
```

---

bounds *Get support of a prior or data distribution*

Description

`bounds()` returns the range of the support of a prior or data distribution.

Usage

```
bounds(dist, ...)

## S4 method for signature 'PointMassPrior'
bounds(dist, ...)

## S4 method for signature 'ContinuousPrior'
bounds(dist, ...)
```

Arguments

`dist`	a univariate `distribution` object
`...`	further optional arguments

Value

`numeric` of length two, `c(lower, upper)`

Examples

```
bounds(PointMassPrior(c(0, .5), c(.3, .7)))
# > 0.3 0.7

bounds(ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4)))
# > 0.2 0.4
```

---

c2 *Query critical values of a design*

Description

Methods to access the stage-two critical values of a TwoStageDesign. c2 returns the stage-two critical value conditional on the stage-one test statistic.

Usage

```
c2(d, x1, ...)

## S4 method for signature 'TwoStageDesign,numeric'
c2(d, x1, ...)

## S4 method for signature 'OneStageDesign,numeric'
c2(d, x1, ...)
```

Arguments

d	design
x1	stage-one test statistic
...	further optional arguments

Value

the critical value function c2 of design d at position x1

See Also

TwoStageDesign, see n for accessing the sample size of a design

Examples

```
design <- TwoStageDesign(
  n1    = 25,
  c1f   = 0,
  c1e   = 2.5,
  n2    = 50,
  c2    = 1.96,
```

```
  order = 7L
)

c2(design, 2.2) # 1.96
c2(design, 3.0) # -Inf
c2(design, -1.0) # Inf

design <- TwoStageDesign(
   n1    = 25,
   c1f   = 0,
   c1e   = 2.5,
   n2    = 50,
   c2    = 1.96,
   order = 7L
)

c2(design, 2.2) # 1.96
c2(design, 3.0) # -Inf
c2(design, -1.0) # Inf
```

---

composite *Score Composition*

Description

composite defines new composite scores by point-wise evaluation of scores in any valid numerical expression.

Usage

```
composite(expr, label = NA_character_)

## S4 method for signature 'CompositeScore,TwoStageDesign'
evaluate(s, design, ...)
```

Arguments

expr	Expression (in curly brackets); must contain at least one score variable; if multiple scores are used, they must either all be conditional or unconditional. Currently, no non-score variables are supported
label	object label (string)
s	object of class CompositeScore
design	object
...	further optional arguments

Value

an object of class `CompositeConditionalScore` or `CompositeUnconditionalScore` depending on the class of the scores used in `expr`

See Also

Scores

Examples

```
ess   <- ExpectedSampleSize(Normal(), PointMassPrior(.4, 1))
power <- Power(Normal(), PointMassPrior(.4, 1))

# linear combination:
composite({ess - 50*power})

# control flow (e.g. for and while loops)
composite({
  res <- 0
  for (i in 1:3) {
     res <- res + ess
  }
  res
})

# functional composition
composite({log(ess)})
cp <- ConditionalPower(Normal(), PointMassPrior(.4, 1))
composite({3*cp})
```

`condition` *Condition a prior on an interval*

Description

Restrict an object of class `Prior` to a sub-interval and re-normalize the PDF.

Usage

```
condition(dist, interval, ...)

## S4 method for signature 'PointMassPrior,numeric'
condition(dist, interval, ...)

## S4 method for signature 'ContinuousPrior,numeric'
condition(dist, interval, ...)
```

Arguments

`dist`	a univariate distribution object
`interval`	length-two numeric vector giving the parameter interval to condition on
`...`	further optional arguments

Value

conditional Prior on given interval

Examples

```
tmp <- condition(PointMassPrior(c(0, .5), c(.3, .7)), c(-1, .25))
expectation(tmp, identity) # 0

tmp <- condition(
    ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4)),
    c(.3, .5)
)
bounds(tmp) # c(.3, .4)
```

ConditionalPower-class

(Conditional) Power of a Design

Description

This score evaluates $\boldsymbol{P}[X_2 > c_2(design, X_1)|X_1 = x_1]$. Note that the distribution of X_2 is the posterior predictive after observing $X_1 = x_1$.

Usage

```
ConditionalPower(dist, prior, label = "Pr[x2>=c2(x1)|x1]")

Power(dist, prior, label = "Pr[x2>=c2(x1)]")

## S4 method for signature 'ConditionalPower,TwoStageDesign'
evaluate(s, design, x1, optimization = FALSE, ...)
```

Arguments

`dist`	a univariate distribution object
`prior`	a Prior object
`label`	object label (string)
`s`	Score object
`design`	object

x1	stage-one test statistic
optimization	logical, if TRUE uses a relaxation to real parameters of the underlying design; used for smooth optimization.
...	further optional arguments

See Also

Scores

Examples

```
prior <- PointMassPrior(.4, 1)
cp <- ConditionalPower(Normal(), prior)
evaluate(
   cp,
   TwoStageDesign(50, .0, 2.0, 50, 2.0, order = 5L),
   x1 = 1
)
# these two are equivalent:
expected(cp, Normal(), prior)
Power(Normal(), prior)
```

ConditionalSampleSize-class

(Conditional) Sample Size of a Design

Description

This score simply evaluates n(d, x1) for a design d and the first-stage outcome x1. The data distribution and prior are only relevant when it is integrated.

Usage

```
ConditionalSampleSize(label = "n(x1)")

ExpectedSampleSize(dist, prior, label = "E[n(x1)]")

## S4 method for signature 'ConditionalSampleSize,TwoStageDesign'
evaluate(s, design, x1, optimization = FALSE, ...)
```

Arguments

label	object label (string)
dist	a univariate distribution object
prior	a Prior object
s	Score object

design	object
x1	stage-one test statistic
optimization	logical, if TRUE uses a relaxation to real parameters of the underlying design; used for smooth optimization.
...	further optional arguments

See Also

Scores

Examples

```
design <- TwoStageDesign(50, .0, 2.0, 50, 2.0, order = 5L)
prior  <- PointMassPrior(.4, 1)

css   <- ConditionalSampleSize()
evaluate(css, design, c(0, .5, 3))

ess   <- ExpectedSampleSize(Normal(), prior)

# those two are equivalent
evaluate(ess, design)
evaluate(expected(css, Normal(), prior), design)
```

Constraints *Formulating Constraints*

Description

Conceptually, constraints work very similar to scores (any score can be put in a constraint). Currently, constraints of the form 'score <=/>= x', 'x <=/>= score' and 'score <=/>= score' are admissible.

Usage

```
## S4 method for signature 'Constraint,TwoStageDesign'
evaluate(s, design, optimization = FALSE, ...)

## S4 method for signature 'ConditionalScore,numeric'
e1 <= e2

## S4 method for signature 'ConditionalScore,numeric'
e1 >= e2

## S4 method for signature 'numeric,ConditionalScore'
e1 <= e2
```

```
## S4 method for signature 'numeric,ConditionalScore'
e1 >= e2

## S4 method for signature 'ConditionalScore,ConditionalScore'
e1 <= e2

## S4 method for signature 'ConditionalScore,ConditionalScore'
e1 >= e2

## S4 method for signature 'UnconditionalScore,numeric'
e1 <= e2

## S4 method for signature 'UnconditionalScore,numeric'
e1 >= e2

## S4 method for signature 'numeric,UnconditionalScore'
e1 <= e2

## S4 method for signature 'numeric,UnconditionalScore'
e1 >= e2

## S4 method for signature 'UnconditionalScore,UnconditionalScore'
e1 <= e2

## S4 method for signature 'UnconditionalScore,UnconditionalScore'
e1 >= e2
```

Arguments

`s`	`Score` object
`design`	object
`optimization`	logical, if `TRUE` uses a relaxation to real parameters of the underlying design; used for smooth optimization.
`...`	further optional arguments
`e1`	left hand side (score or numeric)
`e2`	right hand side (score or numeric)

Value

an object of class `Constraint`

See Also

`minimize`

Examples

```
design <- OneStageDesign(50, 1.96)

cp     <- ConditionalPower(Normal(), PointMassPrior(0.4, 1))
pow    <- Power(Normal(), PointMassPrior(0.4, 1))

# unconditional power constraint
constraint1 <- pow >= 0.8
evaluate(constraint1, design)

# conditional power constraint
constraint2 <- cp  >= 0.7
evaluate(constraint2, design, .5)
constraint3 <- 0.7 <= cp # same as constraint2
evaluate(constraint3, design, .5)
```

ContinuousPrior-class   *Continuous univariate prior distributions*

Description

ContinuousPrior is a sub-class of Prior implementing a generic representation of continuous prior distributions over a compact interval on the real line.

Usage

```
ContinuousPrior(
  pdf,
  support,
  order = 10,
  label = NA_character_,
  tighten_support = FALSE,
  check_normalization = TRUE
)
```

Arguments

pdf	vectorized univariate PDF function
support	numeric vector of length two with the bounds of the compact interval on which the pdf is positive.
order	integer, integration order of the employed Gaussian quadrature integration rule to evaluate scores. Automatically set to length(n2_pivots) if length(n2_pivots) == length(c2_pivots) > 1, otherwise c2 and n2 are taken to be constant in stage-two and replicated to match the number of pivots specified by order
label	object label (string)

`tighten_support`
: logical indicating if the support should be tightened

`check_normalization`
: logical indicating if it should be checked that `pdf` defines a density.

Slots

`pdf` cf. parameter 'pdf'

`support` cf. parameter 'support'

`pivots` normalized pivots for integration rule (in [-1, 1]) the actual pivots are scaled to the support of the prior

`weights` weights of of integration rule at `pivots` for approximating integrals over `delta`

See Also

Discrete priors are supported via `PointMassPrior`

Examples

```
ContinuousPrior(function(x) 2*x, c(0, 1))
```

`cumulative_distribution_function` *Cumulative distribution function*

Description

`cumulative_distribution_function` evaluates the cumulative distribution function of a specific distribution `dist` at a point `x`.

Usage

```
cumulative_distribution_function(dist, x, n, theta, ...)

## S4 method for signature 'Binomial,numeric,numeric,numeric'
cumulative_distribution_function(dist, x, n, theta, ...)

## S4 method for signature 'Normal,numeric,numeric,numeric'
cumulative_distribution_function(dist, x, n, theta, ...)

## S4 method for signature 'Student,numeric,numeric,numeric'
cumulative_distribution_function(dist, x, n, theta, ...)
```

Arguments

`dist`	a univariate `distribution` object
`x`	outcome
`n`	sample size
`theta`	distribution parameter
`...`	further optional arguments

Details

If the distribution is `Binomial`, $theta$ denotes the rate difference between intervention and control group. Then, the mean is assumed to be $\sqrt{n}theta$.

If the distribution is `Normal`, then the mean is assumed to be $\sqrt{n}theta$.

Value

value of the cumulative distribution function at point `x`.

Examples

```
cumulative_distribution_function(Binomial(.1, TRUE), 1, 50, .3)

cumulative_distribution_function(Normal(), 1, 50, .3)

cumulative_distribution_function(Student(two_armed = FALSE), .75, 50, .9)
```

DataDistribution-class   *Data distributions*

Description

`DataDistribution` is an abstract class used to represent the distribution of a sufficient statistic `x` given a sample size `n` and a single parameter value `theta`.

Arguments

`x`	outcome
`n`	sample size
`theta`	distribution parameter
`...`	further optional arguments

Details

This abstraction layer allows the representation of t-distributions (unknown variance), normal distribution (known variance), and normal approximation of a binary endpoint. Currently, the two implemented versions are Normal-class and Binomial-class.

The logical option two_armed allows to decide whether a one-arm or a two-arm (the default) design should be computed. In the case of a two-arm design all sample sizes are per group.

Slots

two_armed Logical that indicates if a two-arm design is assumed.

Examples

```
normaldist   <- Normal(two_armed = FALSE)
binomialdist <- Binomial(rate_control = .25, two_armed = TRUE)
```

---

expectation *Expected value of a function*

Description

Computes the expected value of a vectorized, univariate function f with respect to a distribution dist. I.e., $\boldsymbol{E}\big[f(X)\big]$.

Usage

```
expectation(dist, f, ...)

## S4 method for signature 'PointMassPrior,function'
expectation(dist, f, ...)

## S4 method for signature 'ContinuousPrior,function'
expectation(dist, f, ...)
```

Arguments

dist	a univariate distribution object
f	a univariate function, must be vectorized
...	further optional arguments

Value

numeric, expected value of f with respect to dist

Examples

```
expectation(PointMassPrior(c(0, .5), c(.3, .7)), identity)
# > .35

expectation(
    ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4)),
    identity
)
# > 0.3
```

---

get_initial_design *Initial design*

Description

The optimization method minimize requires an initial design for optimization. The function get_initial_design provides an initial guess based on a fixed design that fulfills constraints on type I error rate and power. Note that a situation-specific initial design may be much more efficient.

Usage

```
get_initial_design(
  theta,
  alpha,
  beta,
  type = c("two-stage", "group-sequential", "one-stage"),
  dist = Normal(),
  order = 7L,
  ...
)
```

Arguments

theta	the alternative effect size in the normal case, the rate difference under the alternative in the binomial case
alpha	maximal type I error rate
beta	maximale type II error rate
type	is a two-stage, group-sequential, or one-stage design requried?
dist	distribution of the test statistic
order	desired integration order
...	further optional arguments

Details

The distribution of the test statistic is specified by dist. The default assumes a two-armed z-test.

Value

An object of class TwoStageDesign.

Examples

```
init <- get_initial_design(
   theta = 0.3,
   alpha = 0.025,
   beta  = 0.2,
   type  = "two-stage",
   dist  = Normal(two_armed = FALSE),
   order = 7L
)
```

---

get_lower_boundary_design
Boundary designs

Description

The optimization method minimize is based on the package nloptr. This requires upper and lower boundaries for optimization. Such boundaries can be computed via lower_boundary_design respectively upper_boundary_design. They are implemented by default in minimize. Note that minimize allows the user to define its own boundary designs, too.

Usage

```
get_lower_boundary_design(initial_design, ...)

get_upper_boundary_design(initial_design, ...)

## S4 method for signature 'OneStageDesign'
get_lower_boundary_design(initial_design, n1 = 1, c1_buffer = 2, ...)

## S4 method for signature 'GroupSequentialDesign'
get_lower_boundary_design(
  initial_design,
  n1 = 1,
  n2_pivots = 1,
  c1_buffer = 2,
  c2_buffer = 2,
  ...
)

## S4 method for signature 'TwoStageDesign'
get_lower_boundary_design(
```

```
  initial_design,
  n1 = 1,
  n2_pivots = 1,
  c1_buffer = 2,
  c2_buffer = 2,
  ...
)

## S4 method for signature 'OneStageDesign'
get_upper_boundary_design(
  initial_design,
  n1 = 5 * initial_design@n1,
  c1_buffer = 2,
  ...
)

## S4 method for signature 'GroupSequentialDesign'
get_upper_boundary_design(
  initial_design,
  n1 = 5 * initial_design@n1,
  n2_pivots = 5 * initial_design@n2_pivots,
  c1_buffer = 2,
  c2_buffer = 2,
  ...
)

## S4 method for signature 'TwoStageDesign'
get_upper_boundary_design(
  initial_design,
  n1 = 5 * initial_design@n1,
  n2_pivots = 5 * initial_design@n2_pivots,
  c1_buffer = 2,
  c2_buffer = 2,
  ...
)
```

Arguments

`initial_design`	The initial design
`...`	optional arguments The values `c1f` and `c1e` from the initial design are shifted to `c1f - c1_buffer` and `c1e - c1_buffer` in `get_lower_boundary_design`, respectively, to `c1f + c1_buffer` and `c1e + c1_buffer` in `get_upper_boundary_design`. This is handled analogously with `c2_pivots` and `c2_buffer`.
`n1`	bound for the first-stage sample size n1
`c1_buffer`	shift of the early-stopping boundaries from the initial ones
`n2_pivots`	bound for the second-stage sample size n2
`c2_buffer`	shift of the final decision boundary from the initial one

Value

An object of class TwoStageDesign.

Examples

```
initial_design <- TwoStageDesign(
  n1    = 25,
  c1f   = 0,
  c1e   = 2.5,
  n2    = 50,
  c2    = 1.96,
  order = 7L
  )
get_lower_boundary_design(initial_design)
```

GroupSequentialDesign-class
Group-sequential two-stage designs

Description

Group-sequential designs are a sub-class of the TwoStageDesign class with constant stage-two sample size. See TwoStageDesign for slot details. Any group-sequential design can be converted to a fully flexible TwoStageDesign (see examples section).

Usage

```
GroupSequentialDesign(n1, c1f, c1e, n2_pivots, c2_pivots, order = NULL, ...)

## S4 method for signature 'GroupSequentialDesign'
TwoStageDesign(n1, ...)
```

Arguments

n1	stage one sample size or GroupSequentialDesign object to convert (overloaded from TwoStageDesign)
c1f	early futility stopping boundary
c1e	early efficacy stopping boundary
n2_pivots	numeric of length one, stage-two sample size
c2_pivots	numeric vector, stage-two critical values on the integration pivot points
order	of the Gaussian uadrature rule to use for integration, set to length(c2_pivots) if NULL, otherwise first value of c2_pivots is repeated 'order'-times.
...	further optional arguments

See Also

TwoStageDesign for superclass and inherited methods

Examples

```
design <- GroupSequentialDesign(25, 0, 2, 25, c(1, 1.5, 2.5))
summary(design)

TwoStageDesign(design)
```

---

make_tunable *Fix parameters during optimization*

Description

The methods make_fixed and make_tunable can be used to modify the 'tunability' status of parameters in a TwoStageDesign object. Tunable parameters are optimized over, non-tunable ('fixed') parameters are considered given and not altered during optimization.

Usage

```
make_tunable(x, ...)

## S4 method for signature 'TwoStageDesign'
make_tunable(x, ...)

make_fixed(x, ...)

## S4 method for signature 'TwoStageDesign'
make_fixed(x, ...)
```

Arguments

x	TwoStageDesign object
...	unquoted names of slots for which the tunability status should be changed.

Value

an updated object of class TwoStageDesign

See Also

TwoStageDesign, tunable_parameters for converting tunable parameters of a design object to a numeric vector (and back), and minimize for the actual minimzation procedure

Examples

```
design <- TwoStageDesign(25, 0, 2, 25, 2, order = 5)
# default: all parameters are tunable (except integration pivots,
# weights and tunability status itself)
design@tunable

# make n1 and the pivots of n2 fixed (not changed during optimization)
design <- make_fixed(design, n1, n2_pivots)
design@tunable

# make them tunable again
design <- make_tunable(design, n1, n2_pivots)
design@tunable
```

MaximumSampleSize-class *Maximum Sample Size of a Design*

Description

This score evaluates `max(n(d))` for a design d.

Usage

```
MaximumSampleSize(label = "max(n(x1))")

## S4 method for signature 'MaximumSampleSize,TwoStageDesign'
evaluate(s, design, optimization = FALSE, ...)
```

Arguments

`label`	object label (string)
`s`	`Score` object
`design`	object
`optimization`	logical, if `TRUE` uses a relaxation to real parameters of the underlying design; used for smooth optimization.
`...`	further optional arguments

See Also

Scores for general scores and ConditionalSampleSize for evaluating the sample size point-wise.

Examples

```
design <- TwoStageDesign(50, .0, 2.0, 50, 2.0, order = 5L)
mss    <- MaximumSampleSize()
evaluate(mss, design)
```

minimize *Find optimal two-stage design by constraint minimization*

Description

minimize takes an unconditional score and a constraint set (or no constraint) and solves the corresponding minimization problem using nloptr (using COBYLA by default). An initial design has to be defined. It is also possible to define lower- and upper-boundary designs. If this is not done, the boundaries are determined automatically heuristically.

Usage

```
minimize(
  objective,
  subject_to,
  initial_design,
  lower_boundary_design = get_lower_boundary_design(initial_design),
  upper_boundary_design = get_upper_boundary_design(initial_design),
  opts = list(algorithm = "NLOPT_LN_COBYLA", xtol_rel = 1e-05, maxeval = 10000),
  ...
)
```

Arguments

objective	objective function
subject_to	constraint collection
initial_design	initial guess (x0 for nloptr)
lower_boundary_design	design specifying the lower boundary.
upper_boundary_design	design specifying the upper boundary
opts	options list passed to nloptr
...	further optional arguments passed to nloptr

Value

a list with elements:

design	The resulting optimal design
nloptr_return	Output of the corresponding nloptr call
call_args	The arguments given to the optimization call

Examples

```
# Define Type one error rate
toer <- Power(Normal(), PointMassPrior(0.0, 1))

# Define Power at delta = 0.4
pow <- Power(Normal(), PointMassPrior(0.4, 1))

# Define expected sample size at delta = 0.4
ess <- ExpectedSampleSize(Normal(), PointMassPrior(0.4, 1))

# Compute design minimizing ess subject to power and toer constraints

minimize(

   ess,

   subject_to(
      toer <= 0.025,
      pow  >= 0.9
   ),

   initial_design = TwoStageDesign(50, .0, 2.0, 60.0, 2.0, 5L)

)
```

---

n1 *Query sample size of a design*

Description

Methods to access the stage-one, stage-two, or overall sample size of a `TwoStageDesign`. `n1` returns the first-stage sample size of a design, `n2` the stage-two sample size conditional on the stage-one test statistic and `n` the overall sample size `n1 + n2`. Internally, objects of the class `TwoStageDesign` allow non-natural, real sample sizes to allow smooth optimization (cf. `minimize` for details). The optional argument `round` allows to switch between the internal real representation and a rounded version (rounding to the next positive integer).

Usage

```
n1(d, ...)

## S4 method for signature 'TwoStageDesign'
n1(d, round = TRUE, ...)

n2(d, x1, ...)
```

```
## S4 method for signature 'TwoStageDesign,numeric'
n2(d, x1, round = TRUE, ...)

n(d, x1, ...)

## S4 method for signature 'TwoStageDesign,numeric'
n(d, x1, round = TRUE, ...)

## S4 method for signature 'GroupSequentialDesign,numeric'
n2(d, x1, round = TRUE, ...)

## S4 method for signature 'OneStageDesign,numeric'
n2(d, x1, ...)
```

Arguments

`d`	design
`...`	further optional arguments
`round`	`logical` should sample sizes be rounded to next integer?
`x1`	stage-one test statistic

Value

sample size value of design `d` at point `x1`

See Also

`TwoStageDesign`, see `c2` for accessing the critical values

Examples

```
design <- TwoStageDesign(
   n1    = 25,
   c1f   = 0,
   c1e   = 2.5,
   n2    = 50,
   c2    = 1.96,
   order = 7L
)

n1(design) # 25
design@n1 # 25

n(design, x1 = 2.2) # 75
```

N1-class *Regularize n1*

Description

N1 is a class that computes the n1 value of a design. This can be used as a score in minimize.

Usage

```
N1(label = NA_character_)

## S4 method for signature 'N1,TwoStageDesign'
evaluate(s, design, optimization = FALSE, ...)
```

Arguments

label	object label (string)
s	Score object
design	object
optimization	logical, if TRUE uses a relaxation to real parameters of the underlying design; used for smooth optimization.
...	further optional arguments

Value

an object of class N1

See Also

See AverageN2 for a regularization of the second-stage sample size.

Examples

```
n1_score <- N1()

evaluate(
   N1(),
   TwoStageDesign(70, 0, 2, rep(60, 6), rep(1.7, 6))
) # 70
```

Normal-class *Normal data distribution*

Description

Implements a normal data distribution for z-values given an observed z-value and stage size. Standard deviation is 1 and mean $\theta\sqrt{n}$ where θ is the standardized effect size. The option `two_armed` can be set to decide whether a one-arm or a two-arm design should be computed.

Usage

```
Normal(two_armed = TRUE)

## S4 method for signature 'Normal'
quantile(x, probs, n, theta, ...)

## S4 method for signature 'Normal,numeric'
simulate(object, nsim, n, theta, seed = NULL, ...)
```

Arguments

`two_armed`	logical indicating if a two-armed trial is regarded
`x`	outcome
`probs`	vector of probabilities
`n`	sample size
`theta`	distribution parameter
`...`	further optional arguments
`object`	object of class `Normal`
`nsim`	number of simulation runs
`seed`	random seed

Details

See `DataDistribution-class` for more details.

See Also

see `probability_density_function` and `cumulative_distribution_function` to evaluate the pdf and the cdf, respectively.

Examples

```
datadist <- Normal(two_armed = TRUE)
```

OneStageDesign-class *One-stage designs*

Description

OneStageDesign implements a one-stage design as special case of a two-stage design, i.e. as subclass of TwoStageDesign. This is possible by defining $n_2 = 0$, $c = c_1^f = c_1^e$, $c_2(x_1) = ifelse(x_1 < c, Inf, -Inf)$. No integration pivots etc are required (set to NaN).

Usage

```
OneStageDesign(n, c)

## S4 method for signature 'OneStageDesign'
TwoStageDesign(n1, order = 5L, eps = 0.01, ...)

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

Arguments

n	sample size (stage-one sample size)
c	rejection boundary ($c = c_1^f = c_1^e$)
n1	OneStageDesign object to convert, overloaded from TwoStageDesign
order	integer >= 2, default is 5; order of Gaussian quadrature integration rule to use for new TwoStageDesign.
eps	numeric > 0, default = .01; the single critical value c must be split in a continuation interval [c1f, c1e]; this is given by c +/- eps.
...	further optional arguments
x	design to plot
y	not used

Details

Note that the default plot,TwoStageDesign-method method is not supported for OneStageDesign objects.

See Also

TwoStageDesign, GroupSequentialDesign

Examples

```
design <- OneStageDesign(30, 1.96)
summary(design)
design <- TwoStageDesign(design)
summary(design)
```

---

plot,TwoStageDesign-method

Plot `TwoStageDesign` *with optional set of conditional scores*

Description

This method allows to plot the stage-two sample size and decision boundary functions of a chosen design.

Usage

```
## S4 method for signature 'TwoStageDesign'
plot(x, y = NULL, ..., rounded = TRUE, k = 100)
```

Arguments

`x`	design to plot
`y`	not used
`...`	further named `ConditinonalScores` to plot for the design and/or further graphic parameters
`rounded`	should n-values be rounded?
`k`	number of points to use for plotting

Details

`TwoStageDesign` and user-defined elements of the class `ConditionalScore`.

Value

a plot of the two-stage design

See Also

`TwoStageDesign`

Examples

```
design <- TwoStageDesign(50, 0, 2, 50, 2, 5)
cp     <- ConditionalPower(dist = Normal(), prior = PointMassPrior(.4, 1))
plot(design, "Conditional Power" = cp, cex.axis = 2)
```

PointMassPrior-class *Univariate discrete point mass priors*

Description

PointMassPrior is a sub-class of Prior representing a univariate prior over a discrete set of points with positive probability mass.

Usage

```
PointMassPrior(theta, mass, label = NA_character_)
```

Arguments

theta	numeric vector of pivot points with positive prior mass
mass	numeric vector of probability masses at the pivot points (must sum to 1)
label	object label (string)

Value

an object of class PointMassPrior, theta is automatically sorted in ascending order

Slots

theta cf. parameter 'theta'

mass cf. parameter 'mass'

See Also

To represent continuous prior distributions use ContinuousPrior.

Examples

```
PointMassPrior(c(0, .5), c(.3, .7))
```

posterior *Compute posterior distribution*

Description

Return posterior distribution given observing stage-one outcome.

Usage

```
posterior(dist, prior, x1, n1, ...)

## S4 method for signature 'DataDistribution,PointMassPrior,numeric'
posterior(dist, prior, x1, n1, ...)

## S4 method for signature 'DataDistribution,ContinuousPrior,numeric'
posterior(dist, prior, x1, n1, ...)
```

Arguments

`dist`	a univariate `distribution` object
`prior`	a `Prior` object
`x1`	stage-one test statistic
`n1`	stage-one sample size
`...`	further optional arguments

Value

Object of class `Prior`

Examples

```
posterior(Normal(), PointMassPrior(0, 1), 2, 20)

tmp <- ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4))
posterior(Normal(), tmp, 2, 20)
```

predictive_cdf *Predictive CDF*

Description

predictive_cdf() evaluates the predictive CDF of the model specified by a DataDistribution dist and Prior at the given stage-one outcome.

Usage

```
predictive_cdf(dist, prior, x1, n1, ...)

## S4 method for signature 'DataDistribution,PointMassPrior,numeric'
predictive_cdf(dist, prior, x1, n1, ...)

## S4 method for signature 'DataDistribution,ContinuousPrior,numeric'
predictive_cdf(
  dist,
  prior,
  x1,
  n1,
  k = 10 * (prior@support[2] - prior@support[1]) + 1,
  ...
)
```

Arguments

dist	a univariate distribution object
prior	a Prior object
x1	stage-one test statistic
n1	stage-one sample size
...	further optional arguments
k	number of pivots for crude integral approximation

Value

numeric, value of the predictive CDF

Examples

```
predictive_cdf(Normal(), PointMassPrior(.0, 1), 0, 20) # .5

tmp <- ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4))
predictive_cdf(Normal(), tmp, 2, 20)
```

predictive_pdf *Predictive PDF*

Description

predictive_pdf() evaluates the predictive PDF of the model specified by a DataDistribution dist and Prior at the given stage-one outcome.

Usage

```
predictive_pdf(dist, prior, x1, n1, ...)

## S4 method for signature 'DataDistribution,PointMassPrior,numeric'
predictive_pdf(dist, prior, x1, n1, ...)

## S4 method for signature 'DataDistribution,ContinuousPrior,numeric'
predictive_pdf(
  dist,
  prior,
  x1,
  n1,
  k = 10 * (prior@support[2] - prior@support[1]) + 1,
  ...
)
```

Arguments

dist	a univariate distribution object
prior	a Prior object
x1	stage-one test statistic
n1	stage-one sample size
...	further optional arguments
k	number of pivots for crude integral approximation

Value

numeric, value of the predictive PDF

Examples

```
predictive_pdf(Normal(), PointMassPrior(.3, 1), 1.5, 20) # ~.343

tmp <- ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4))
predictive_pdf(Normal(), tmp, 2, 20)
```

print.adoptrOptimizationResult   *Printing an optimization result*

Description

Printing an optimization result

Usage

```
print(x, ...)
```

Arguments

x	object to print
...	further arguments passed form other methods

Prior-class   *Univariate prior on model parameter*

Description

A `Prior` object represents a prior distribution on the single model parameter of a `DataDistribution` class object. Together a prior and data-distribution specify the class of the joint distribution of the test statisic, X, and its parameter, theta. Currently, **adoptr** only allows simple models with a single parameter. Implementations for PointMassPrior and ContinuousPrior are available.

Details

For an example on working with priors, see here.

See Also

For the available methods, see `bounds`, `expectation`, `condition`, `predictive_pdf`, `predictive_cdf`, `posterior`

Examples

```
disc_prior <- PointMassPrior(c(0.1, 0.25), c(0.4, 0.6))

cont_prior <- ContinuousPrior(
  pdf     = function(x) dnorm(x, mean = 0.3, sd = 0.2),
  support = c(-2, 3)
)
```

probability_density_function

Probability density function

Description

probability_density_function evaluates the probability density function of a specific distribution dist at a point x.

Usage

```
probability_density_function(dist, x, n, theta, ...)

## S4 method for signature 'Binomial,numeric,numeric,numeric'
probability_density_function(dist, x, n, theta, ...)

## S4 method for signature 'Normal,numeric,numeric,numeric'
probability_density_function(dist, x, n, theta, ...)

## S4 method for signature 'Student,numeric,numeric,numeric'
probability_density_function(dist, x, n, theta, ...)
```

Arguments

dist	a univariate distribution object
x	outcome
n	sample size
theta	distribution parameter
...	further optional arguments

Details

If the distribution is Binomial, $theta$ denotes the rate difference between intervention and control group. Then, the mean is assumed to be $\sqrt{n}theta$.

If the distribution is Normal, then the mean is assumed to be $\sqrt{n}theta$.

Value

value of the probability density function at point x.

Examples

```
probability_density_function(Binomial(.2, FALSE), 1, 50, .3)

probability_density_function(Normal(), 1, 50, .3)

probability_density_function(Student(TRUE), 1, 40, 1.1)
```

Scores *Scores*

Description

In adoptr scores are used to assess the performance of a design. This can be done either conditionally on the observed stage-one outcome or unconditionally. Consequently, score objects are either of class ConditionalScore or UnconditionalScore.

Usage

```
expected(s, data_distribution, prior, ...)

## S4 method for signature 'ConditionalScore'
expected(s, data_distribution, prior, label = NA_character_, ...)

evaluate(s, design, ...)

## S4 method for signature 'IntegralScore,TwoStageDesign'
evaluate(s, design, optimization = FALSE, subdivisions = 10000L, ...)
```

Arguments

s	Score object
data_distribution	DataDistribution object
prior	a Prior object
...	further optional arguments
label	object label (string)
design	object
optimization	logical, if TRUE uses a relaxation to real parameters of the underlying design; used for smooth optimization.
subdivisions	maximal number of subdivisions when evaluating an integral score using adaptive quadrature (optimization = FALSE)

Details

All scores can be evaluated on a design using the evaluate method. Note that evaluate requires a third argument x1 for conditional scores (observed stage-one outcome). Any ConditionalScore can be converted to a UnconditionalScore by forming its expected value using expected. The returned unconditional score is of class IntegralScore.

Value

No return value. Generic description of class Score.

See Also

ConditionalPower, ConditionalSampleSize, composite

Examples

```
design <- TwoStageDesign(
  n1    = 25,
  c1f   = 0,
  c1e   = 2.5,
  n2    = 50,
  c2    = 1.96,
  order = 7L
)
prior <- PointMassPrior(.3, 1)

# conditional
cp <- ConditionalPower(Normal(), prior)
expected(cp, Normal(), prior)
evaluate(cp, design, x1 = .5)

# unconditional
power <- Power(Normal(), prior)
evaluate(power, design)
evaluate(power, design, optimization = TRUE) # use non-adaptive quadrature
```

---

simulate,TwoStageDesign,numeric-method

Draw samples from a two-stage design

Description

`simulate` allows to draw samples from a given TwoStageDesign.

Usage

```
## S4 method for signature 'TwoStageDesign,numeric'
simulate(object, nsim, dist, theta, seed = NULL, ...)
```

Arguments

object	TwoStageDesign to draw samples from
nsim	number of simulation runs
dist	data distribution
theta	location parameter of the data distribution
seed	random seed
...	further optional arguments

Value

`simulate()` returns a `data.frame` with `nsim` rows and for each row (each simulation run) the following columns

- theta: The effect size
- n1: First-stage sample size
- c1f: Stopping for futility boundary
- c1e: Stopping for efficacy boundary
- x1: First-stage outcome
- n2: Resulting second-stage sample size after observing x1
- c2: Resulting second-stage decision-boundary after observing x1
- x2: Second-stage outcome
- reject: Decision whether the null hypothesis is rejected or not

See Also

`TwoStageDesign`

Examples

```
design <- TwoStageDesign(25, 0, 2, 25, 2, order = 5)
# draw samples assuming two-armed design
simulate(design, 10, Normal(), .3, 42)
```

Student-class *Student's t data distribution*

Description

Implements exact t-distributions instead of a normal approximation

Usage

```
Student(two_armed = TRUE)

## S4 method for signature 'Student'
quantile(x, probs, n, theta, ...)

## S4 method for signature 'Student,numeric'
simulate(object, nsim, n, theta, seed = NULL, ...)
```

Arguments

`two_armed`	logical indicating if a two-armed trial is regarded
`x`	outcome
`probs`	vector of probabilities
`n`	sample size
`theta`	distribution parameter
`...`	further optional arguments
`object`	object of class `Student`
`nsim`	number of simulation runs
`seed`	random seed

See Also

see `probability_density_function` and `cumulative_distribution_function` to evaluate the pdf and the cdf, respectively.

Examples

```
datadist <- Student(two_armed = TRUE)
```

---

subject_to — *Create a collection of constraints*

Description

`subject_to(...)` can be used to generate an object of class `ConstraintsCollection` from an arbitrary number of (un)conditional constraints.

Usage

```
subject_to(...)

## S4 method for signature 'ConstraintsCollection,TwoStageDesign'
evaluate(s, design, optimization = FALSE, ...)
```

Arguments

`...`	either constraint objects (for `subject_to` or optional arguments passed to `evaluate`)
`s`	object of class `ConstraintCollection`
`design`	object
`optimization`	logical, if `TRUE` uses a relaxation to real parameters of the underlying design; used for smooth optimization.

Value

an object of class ConstraintsCollection

See Also

subject_to is intended to be used for constraint specification the constraints in minimize.

Examples

```
# define type one error rate and power
toer  <- Power(Normal(), PointMassPrior(0.0, 1))
power <- Power(Normal(), PointMassPrior(0.4, 1))

# create constrain collection
subject_to(
  toer  <= 0.025,
  power >= 0.9
)
```

---

tunable_parameters *Switch between numeric and S4 class representation of a design*

Description

Get tunable parameters of a design as numeric vector via tunable_parameters or update a design object with a suitable vector of values for its tunable parameters.

Usage

```
tunable_parameters(object, ...)

## S4 method for signature 'TwoStageDesign'
tunable_parameters(object, ...)

## S4 method for signature 'TwoStageDesign'
update(object, params, ...)

## S4 method for signature 'OneStageDesign'
update(object, params, ...)
```

Arguments

object	TwoStageDesign object to update
...	further optional arguments
params	vector of design parameters, must be in same order as returned by tunable_parameters

Details

The `tunable` slot of a `TwoStageDesign` stores information about the set of design parameters which are considered fixed (not changed during optimization) or tunable (changed during optimization). For details on how to fix certain parameters or how to make them tunable again, see `make_fixed` and `make_tunable`.

Value

`tunable_parameters` returns the numerical values of all tunable parameters as a vector. `update` returns the updated design.

See Also

`TwoStageDesign`

Examples

```
design  <- TwoStageDesign(25, 0, 2, 25, 2, order = 5)
tunable_parameters(design)
design2 <- update(design, tunable_parameters(design) + 1)
tunable_parameters(design2)
```

TwoStageDesign-class *Two-stage designs*

Description

TwoStageDesign is the fundamental design class of the **adoptr** package. Formally, we represent a generic two-stage design as a five-tuple $\left(n_1, c_1^f, c_1^e, n_2(\cdot), c_2(\cdot)\right)$. Here, n_1 is the first-stage sample size (per group), c_1^f and c_1^e are boundaries for early stopping for futility and efficacy, respectively. Since the trial design is a two-stage design, the elements $n_2(\cdot)$ (stage-two sample size) and $c_2(\cdot)$ (stage-two critical value) are functions of the first-stage outcome $X_1 = x_1$. X_1 denotes the first-stage test statistic. A brief description on this definition of two-stage designs can be read here. For available methods, see the 'See Also' section at the end of this page.

Usage

```
TwoStageDesign(n1, ...)

## S4 method for signature 'numeric'
TwoStageDesign(n1, c1f, c1e, n2_pivots, c2_pivots, order = NULL, ...)

## S4 method for signature 'TwoStageDesign'
summary(object, ..., rounded = TRUE)
```

Arguments

n1	stage-one sample size
...	further optional arguments
c1f	early futility stopping boundary
c1e	early efficacy stopping boundary
n2_pivots	numeric vector, stage-two sample size on the integration pivot points
c2_pivots	numeric vector, stage-two critical values on the integration pivot points
order	integer, integration order of the employed Gaussian quadrature integration rule to evaluate scores. Automatically set to length(n2_pivots) if length(n2_pivots) == length(c2_pivots) > 1, otherwise c2 and n2 are taken to be constant in stage-two and replicated to match the number of pivots specified by order
object	object to show
rounded	should rounded n-values be used?

Details

summary can be used to quickly compute and display basic facts about a TwoStageDesign. An arbitrary number of names UnconditionalScore objects can be provided via the optional arguments ... and are included in the summary displayed using print.

Slots

n1 cf. parameter 'n1'

c1f cf. parameter 'c1f'

c1e cf. parameter 'c1e'

n2_pivots vector of length 'order' giving the values of n2 at the pivot points of the numeric integration rule

c2_pivots vector of length order giving the values of c2 at the pivot points of the numeric integration rule

x1_norm_pivots normalized pivots for integration rule (in [-1, 1]) the actual pivots are scaled to the interval [c1f, c1e] and can be obtained by the internal method adoptr:::scaled_integration_pivots(design)

weights weights of of integration rule at x1_norm_pivots for approximating integrals over x1

tunable named logical vector indicating whether corresponding slot is considered a tunable parameter (i.e. whether it can be changed during optimization via minimize or not; cf. make_fixed)

See Also

For accessing sample sizes and critical values safely, see methods in n and c2; for modifying behaviour during optimizaton see make_tunable; to convert between S4 class represenation and numeric vector, see tunable_parameters; for simulating from a given design, see simulate; for plotting see plot,TwoStageDesign-method. Both group-sequential and one-stage designs (!) are implemented as subclasses of TwoStageDesign.

Examples

```
design <- TwoStageDesign(50, 0, 2, 50.0, 2.0, 5)
pow    <- Power(Normal(), PointMassPrior(.4, 1))
summary(design, "Power" = pow)
```

Index