Package ‘adoptr’

January 9, 2020

Type Package

Title Adaptive Optimal Two-Stage Designs in R

Version 0.3.2

Description Optimize one or two-arm, two-stage designs for clinical trials with respect to several pre-implemented objective criteria or implement custom objectives. Optimization under uncertainty and conditional (given stage-one outcome) constraints are supported.


License MIT + file LICENSE

Encoding UTF-8

LazyData true

Suggests knitr, rmarkdown, testthat, covr, rpact, vdiffr

Imports nloptr, methods, glue

VignetteBuilder knitr


RoxygenNote 7.0.2

BugReports https://github.com/kkmann/adoptr/issues

URL https://github.com/kkmann/adoptr

NeedsCompilation no

Author Kevin Kunzmann [aut, cre], Maximilian Pilz [aut]

Maintainer Kevin Kunzmann <kevin.kunzmann@mrc-bsu.cam.ac.uk>

Repository CRAN

Date/Publication 2020-01-09 14:20:14 UTC
R topics documented:

adoptr .................................................. 3
AverageN2-class ...................................... 3
Binomial-class ........................................ 4
bounds .................................................. 5
c2 ....................................................... 6
composite .............................................. 7
condition ................................................ 8
ConditionalPower-class ......................... 9
ConditionalSampleSize-class ........... 10
Constraints ............................................. 11
ContinuousPrior-class ..................... 13
cumulative_distribution_function ........ 14
Data Distribution-class .................... 15
expectation ............................................ 16
get_initial_design .................................. 17
get_lower_boundary_design ...................... 18
GroupSequentialDesign-class ............ 20
make_tunable ......................................... 21
minimize ............................................... 22
n1 ....................................................... 23
N1-class ................................................. 25
Normal-class ......................................... 26
OneStageDesign-class ......................... 27
plot, TwoStageDesign-method ............. 28
PointMassPrior-class ......................... 29
posterior ............................................... 30
predictive_cdf ........................................ 31
predictive_pdf ........................................ 32
print.adoptrOptimizationResult .......... 33
Prior-class ........................................... 33
probability_density_function ............. 34
Scores ................................................... 35
simulate, TwoStageDesign, numeric-method 36
subject_to ............................................ 37
tunable_parameters ............................... 38
TwoStageDesign-class ....................... 39

Index 42
The `adoptr` package provides functionality to explore custom optimal two-stage designs for one- or two-arm superiority tests. Currently, only (asymptotically) normal test statistics are supported. For more details on the theoretical background see <doi:10.1002/sim.8291>. `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 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`.

---

### 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 uniform prior on the continuation region.

**Usage**

```r
AverageN2(label = NA_character_)
```

```r
## 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

```r
avn2 <- AverageN2()

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

Binomial-class

### Description

Implements the normal approximation for a test on rates. The response rate in the control group, \( r_C \), has to be specified by `rate_control`. The null hypothesis is: \( r_E \leq r_C \), where \( r_E \) denotes the response rate in the intervention group. It is tested against the alternative \( r_E > r_C \). The test statistic is given as \( X_1 = (r_E - r_C) / \sqrt{2r_0(1 - r_0)} \), 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.

Usage

```r
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, ...)
```
**bounds**

**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**

All priors have to be defined for the rate difference $r_E - r_C$.

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**

```r
datadist <- Binomial(rate.control = 0.2, two.armed = FALSE)
```

**Description**

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

**Usage**

```r
bounds(dist, ...)
```

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

```r
## 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

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)
condition

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

condition(dist, interval, ...)  

---

**Description**

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

**Usage**

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

```r
## 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

```r
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 \( P[X_2 > c_2(\text{design}, X_1)|X_1 = x_1] \). Note that the distribution of \( X_2 \) is the posterior predictive after observing \( X_1 = x_1 \).

### Usage

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

```r
## 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

```r
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 \(n(d,x_1)\) for a design \(d\) and the first-stage outcome \(x_1\). 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>object label (string)</td>
</tr>
<tr>
<td>dist</td>
<td>a univariate distribution object</td>
</tr>
<tr>
<td>prior</td>
<td>a Prior object</td>
</tr>
<tr>
<td>s</td>
<td>Score object</td>
</tr>
<tr>
<td>design</td>
<td>object</td>
</tr>
<tr>
<td>x1</td>
<td>stage-one test statistic</td>
</tr>
<tr>
<td>optimization</td>
<td>logical, if TRUE uses a relaxation to real parameters of the underlying design; used for smooth optimization.</td>
</tr>
</tbody>
</table>

... 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
## Constraints

### 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)

### See Also

- **minimize**
Examples

design <- OneStageDesign(50, 1.96)

<ct>cp</ct>  <- ConditionalPower(Normal(), PointMassPrior(0.4, 1))
<ct>pow</ct> <- 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)

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdf</td>
<td>vectorized univariate PDF function</td>
</tr>
<tr>
<td>support</td>
<td>numeric vector of length two with the bounds of the compact interval on which the pdf is positive.</td>
</tr>
<tr>
<td>order</td>
<td>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) &gt; 1, otherwise c2 and n2 are taken to be constant in stage-two and replicated to match the number of pivots specified by order</td>
</tr>
<tr>
<td>label</td>
<td>object label (string)</td>
</tr>
</tbody>
</table>
cumulative_distribution_function

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

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 'Normal,numeric,numeric,numeric'
  cumulative_distribution_function(dist, x, n, theta, ...)

  ## S4 method for signature 'Binomial,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 **Normal**, then the mean is assumed to be $\sqrt{n}\theta$.

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

Examples

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

---

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

```r
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 \( \text{dist} \). I.e., \( E[f(X)] \).

**Usage**

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

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

```r
## 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 \( \text{dist} \)

**Examples**

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

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

---

**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**

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

**Arguments**

- `theta` the alternative effect size  
- `alpha` maximal type I error rate  
- `beta` maximal type II error rate  
- `type` is a two-stage, group-sequential, or one-stage design required?  
- `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.

**Examples**

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

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

Usage

\begin{verbatim}
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,
  ...
)
\end{verbatim}
get_lower_boundary_design

---

```r
## 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,
  ...
)
```

```r
## 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
- **n1** bound for the first-stage sample size
- **c1_buffer** shift of the early-stopping boundaries from the initial ones
- **n2_pivots** bound for the second-stage sample size
- **c2_buffer** shift of the final decision boundary from the initial one

Examples

```r
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 quadrature 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)
**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**

```r
make_tunable(x, ...)
```

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

```r
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.

**See Also**

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

**Examples**

```r
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
```
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

```r
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
## Not run:
minimize(
  ess,
  subject_to(
    toer <= 0.025,
    pow >= 0.9
  ),
  initial_design = TwoStageDesign(50, .0, 2.0, 60.0, 2.0, 2L)
)
## End(Not run)

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 '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

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
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 $\theta$ 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 $n_2 = 0, c = c_1^f = c_1^e, c_2(x_1) = \text{ifelse}(x_1 < c, \inf, -\inf)$. 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 $\geq 2$, 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 $[c_1^f, c_1^e]$; this is given by $c \pm \text{eps}$.

... 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 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 ConditionalScores 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.

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)
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

```r
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

```r
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

**Description**

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

**Usage**

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

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

```r
## 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**

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

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

```r
predictive_pdf(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
- `k` number of pivots for crude integral approximation

### Value

numeric, value of the predictive PDF

### Examples

```r
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 statistic, 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))
Description

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

Usage

```r
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 'Binomial,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 `Normal`, then the mean is assumed to be $\sqrt{n}\theta$.

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

Examples

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

Description

In adaptr 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'
evaluated(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.

See Also

ConditionalPower, ConditionalSampleSize, composite
Examples

```r
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**

```r
## 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 \( n_{\text{sim}} \) rows and for each row (each simulation run) the following columns

- \( \theta \): The effect size
- \( n_1 \): First-stage sample size
- \( c_{1f} \): Stopping for futility boundary
- \( c_{1e} \): Stopping for efficacy boundary
- \( x_1 \): First-stage outcome
- \( n_2 \): Resulting second-stage sample size after observing \( x_1 \)
- \( c_{2} \): Resulting second-stage decision-boundary after observing \( x_1 \)
- \( x_2 \): Second-stage outcome
- \( \text{reject} \): 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)

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

```r
# 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, ...)`

```r
## 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`.

See Also

`TwoStageDesign`

Examples

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

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 optimization see make_tunable; to convert between S4 class representation 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

<=, ConditionalScore, ConditionalScore-method (Constraints), 11
<=, ConditionalScore, numeric-method (Constraints), 11
<=, UnconditionalScore, UnconditionalScore-method (Constraints), 11
<=, UnconditionalScore, numeric-method (Constraints), 11
<=, numeric, ConditionalScore-method (Constraints), 11
<=, numeric, UnconditionalScore-method (Constraints), 11
>=, ConditionalScore, ConditionalScore-method (Constraints), 11
>=, ConditionalScore, numeric-method (Constraints), 11
>=, UnconditionalScore, UnconditionalScore-method (Constraints), 11
>=, UnconditionalScore, numeric-method (Constraints), 11
>=, numeric, ConditionalScore-method (Constraints), 11
>=, numeric, UnconditionalScore-method (Constraints), 11

adoptr, 3, 39
AverageN2, 4, 25
AverageN2 (AverageN2-class), 3
AverageN2-class, 3

Binomial, 15, 34
Binomial (Binomial-class), 4
Binomial-class, 4
bounds, 5, 33
bounds, ContinuousPrior-method (bounds), 5
bounds, PointMassPrior-method (bounds), 5
c2, 6, 24, 40
c2, OneStageDesign, numeric-method (c2), 6
c2, TwoStageDesign, numeric-method (c2), 6
composites, 7, 33
condition, 8, 33
condition, ContinuousPrior, numeric-method (condition), 8
condition, PointMassPrior, numeric-method (condition), 8
ConditionalPower, 3, 35
ConditionalPower (ConditionalPower-class), 9
ConditionalPower-class, 9
ConditionalSampleSize, 3, 35
ConditionalSampleSize (ConditionalSampleSize-class), 10
ConditionalSampleSize-class, 10
ConditionalScore, 28
ConstraintCollection (subject_to), 37
Constraints, 11
ContinuousPrior, 3, 29, 33
ContinuousPrior (ContinuousPrior-class), 13
ContinuousPrior-class, 13
cumulative_distribution_function, 5, 14, 26
cumulative_distribution_function, Binomial, numeric, numeric, numeric-method (cumulative_distribution_function), 14
cumulative_distribution_function, Normal, numeric, numeric, numeric-method (cumulative_distribution_function), 14

DataDistribution, 3, 31–33, 35
DataDistribution (DataDistribution-class), 15
DataDistribution-class, 15
distribution, 6, 9–11, 14, 16, 30–32, 34
evaluate (Scores), 35
evaluate, AverageN2, TwoStageDesign-method (AverageN2-class), 3
evaluate, CompositeScore, TwoStageDesign-method (Composite), 7
evaluate, ConditionalPower, TwoStageDesign-method (ConditionalPower-class), 9
evaluate, ConditionalSampleSize, TwoStageDesign-method (ConditionalSampleSize-class), 10
evaluate, Constraint, TwoStageDesign-method (Constraints), 11
evaluate, ConstraintsCollection, TwoStageDesign-method (subject to), 37
evaluate, IntegralScore, TwoStageDesign-method (Scores), 35
evaluate, N1, TwoStageDesign-method (N1-class), 25
expectation, 16, 33
expectation, ContinuousPrior, function-method (expectation), 16
expected (Scores), 35
expected, ConditionalScore-method (Scores), 35
ExpectedSampleSize, 3
ExpectedSampleSize (ConditionalSampleSize-class), 10
get_initial_design, 17
get_lower_boundary_design, 18
get_lower_boundary_design, GroupSequentialDesign-method (get_lower_boundary_design), 18
get_lower_boundary_design, OneStageDesign-method (get_lower_boundary_design), 18
get_lower_boundary_design, TwoStageDesign-method (get_lower_boundary_design), 18
get_upper_boundary_design (get_lower_boundary_design), 18
get_upper_boundary_design, GroupSequentialDesign-method (get_lower_boundary_design), 18
get_upper_boundary_design, OneStageDesign-method (get_lower_boundary_design), 18
get_upperboundary_design, TwoStageDesign-method (get_lower_boundary_design), 18
group-sequential, 40
GroupSequentialDesign, 3, 27
GroupSequentialDesign (GroupSequentialDesign-class), 20
GroupSequentialDesign-class, 20
make_fixed, 39, 40
make_fixed (make_tunable), 21
make_fixed, TwoStageDesign-method (make_tunable), 21
make_tunable, 21, 39, 40
make_tunable, TwoStageDesign-method (make_tunable), 21
minimize, 12, 17, 18, 21, 22, 23, 25, 38, 40
n, 6, 40
n (n1), 23
n, TwoStageDesign, numeric-method (n1), 23
N1, 4, 25
N1 (N1-class), 25
n1, 23
n1, TwoStageDesign-method (n1), 23
N1-class, 25
n2 (n1), 23
n2, GroupSequentialDesign, numeric-method (n1), 23
n2, OneStageDesign, numeric-method (n1), 23
n2, TwoStageDesign, numeric-method (n1), 23
nloptr, 22
Normal, 3, 15, 34
Normal (Normal-class), 26
Normal-class, 26
One-stage designs, 40
OneStageDesign, 3
OneStageDesign (OneStageDesign-class), 27
OneStageDesign-class, 27
plot, OneStageDesign-method (OneStageDesign-class), 27
plot
PointMassPrior, 3, 14, 33
PointMassPrior (PointMassPrior-class), 29
PointMassPrior-class, 29
posterior, 30, 33
posterior, DataDistribution, ContinuousPrior, numeric-method (posterior), 30
INDEX

posterior, DataDistribution, PointMassPrior, numeric-method
   (posterior), 30
Power, 3
Power (ConditionalPower-class), 9
predictive_cdf, 31, 33
predictive_cdf, DataDistribution, ContinuousPrior, numeric-method
   (predictive_cdf), 31
predictive_cdf, DataDistribution, PointMassPrior, numeric-method
   (predictive_cdf), 31
predictive_pdf, 32, 33
predictive_pdf, DataDistribution, ContinuousPrior, numeric-method
   (predictive_pdf), 32
predictive_pdf, DataDistribution, PointMassPrior, numeric-method
   (predictive_pdf), 32
print, 40
print (print.adoptrOptimizationResult), 33
print.adoptrOptimizationResult, 33
Prior, 8–11, 13, 29–32, 35
Prior (Prior-class), 33
Prior-class, 33
probability_density_function, 5, 26, 34
probability_density_function, Binomial, numeric, numeric, numeric-method
   (probability_density_function), 34
probability_density_function, Normal, numeric, numeric, numeric-method
   (probability_density_function), 34
quantile, Binomial-method
   (Binomial-class), 4
quantile, Normal-method (Normal-class), 26
Score, 4, 10–12, 25, 35
Scores, 3, 8, 10, 11, 35
simulate, 40
simulate, Binomial, numeric-method
   (Binomial-class), 4
simulate, Normal, numeric-method
   (Normal-class), 26
simulate, TwoStageDesign, numeric-method, 36
subject_to, 37
summary, TwoStageDesign-method
   (TwoStageDesign-class), 39
tunable_parameters, 21, 38, 40

TwoStageDesign, 3, 6, 20, 21, 23, 24, 27, 28, 36, 37, 39
TwoStageDesign (TwoStageDesign-class),
TwoStageDesign, GroupSequentialDesign-method
   (GroupSequentialDesign-class), 20
TwoStageDesign, numeric-method
   (TwoStageDesign-class), 39
TwoStageDesign, OneStageDesign-method
   (OneStageDesign-class), 27
TwoStageDesign-class, 39
UnconditionalScore, 40
update, OneStageDesign-method
   (tunable_parameters), 38
update, TwoStageDesign-method
   (tunable_parameters), 38

probability_density_function, Binomial, numeric, numeric, numeric-method
   (probability_density_function), 34

quantile, Binomial-method
   (Binomial-class), 4
quantile, Normal-method (Normal-class), 26
Score, 4, 10–12, 25, 35
Scores, 3, 8, 10, 11, 35
simulate, 40
simulate, Binomial, numeric-method
   (Binomial-class), 4
simulate, Normal, numeric-method
   (Normal-class), 26
simulate, TwoStageDesign, numeric-method, 36
subject_to, 37
summary, TwoStageDesign-method
   (TwoStageDesign-class), 39
tunable_parameters, 21, 38, 40