

# Package ‘adoptr’

May 7, 2026

**Type** Package

**Title** Adaptive Optimal Two-Stage Designs

**Version** 1.1.2

**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](https://doi.org/10.1002/sim.8291)> and Kunzmann et al. (2021) <[doi:10.18637/jss.v098.i09](https://doi.org/10.18637/jss.v098.i09)> for details.

**License** MIT + file LICENSE

**Encoding** UTF-8

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

**Imports** nloptr, methods, glue

**VignetteBuilder** knitr

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

**BugReports** <https://github.com/optad/adoptr/issues>

**URL** <https://github.com/optad/adoptr>, <https://optad.github.io/adoptr/>

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Kevin Kunzmann [aut, cph] (ORCID:  
<<https://orcid.org/0000-0002-1140-7143>>),  
Maximilian Pilz [aut, cre] (ORCID:  
<<https://orcid.org/0000-0002-9685-1613>>),

Jan Meis [aut] (ORCID: <<https://orcid.org/0000-0001-5407-7220>>),  
 Nico Bruder [aut] (ORCID: <<https://orcid.org/0009-0004-9522-2075>>)

**Maintainer** Maximilian Pilz <[maximilian.pilz@th-nuernberg.de](mailto:maximilian.pilz@th-nuernberg.de)>

**Repository** CRAN

**Date/Publication** 2026-05-03 13:10:02 UTC

## Contents

adoptr . . . . .	3
ANOVA-class . . . . .	4
AverageN2-class . . . . .	5
Binomial-class . . . . .	6
bounds . . . . .	7
c2 . . . . .	8
ChiSquared-class . . . . .	9
composite . . . . .	10
condition . . . . .	11
ConditionalPower-class . . . . .	12
ConditionalSampleSize-class . . . . .	13
Constraints . . . . .	14
ContinuousPrior-class . . . . .	16
cumulative_distribution_function . . . . .	17
DataDistribution-class . . . . .	18
expectation . . . . .	19
get_initial_design . . . . .	20
get_lower_boundary_design . . . . .	22
GroupSequentialDesign-class . . . . .	24
GroupSequentialDesignSurvival-class . . . . .	25
make_tunable . . . . .	25
MaximumSampleSize-class . . . . .	26
minimize . . . . .	27
n1 . . . . .	29
N1-class . . . . .	30
NestedModels-class . . . . .	31
Normal-class . . . . .	32
OneStageDesign-class . . . . .	33
OneStageDesignSurvival-class . . . . .	35
Pearson2xK-class . . . . .	35
plot,TwoStageDesign-method . . . . .	36
PointMassPrior-class . . . . .	37
posterior . . . . .	38
predictive_cdf . . . . .	39
predictive_pdf . . . . .	40
print.adoptrOptimizationResult . . . . .	41
Prior-class . . . . .	41
probability_density_function . . . . .	42
Scores . . . . .	43

simulate,TwoStageDesign,numeric-method . . . . .	45
Student-class . . . . .	46
subject_to . . . . .	47
Survival-class . . . . .	48
SurvivalDesign . . . . .	49
tunable_parameters . . . . .	50
TwoStageDesign-class . . . . .	51
TwoStageDesignSurvival-class . . . . .	53
ZSquared-class . . . . .	54

<b>Index</b>	<b>55</b>
--------------	-----------

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](https://doi.org/10.1002/sim.8291) and [doi:10.18637/jss.v098.i09](https://doi.org/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](https://doi.org/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

The implemented data distributions are [Normal](#), [Binomial](#), [Student](#), [Survival](#), [ChiSquared](#) (including [Pearson2xK](#) and [ZSquared](#)) and [ANOVA](#).

### 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@th-nuernberg.de> ([ORCID](#))

Authors:

- Maximilian Pilz <maximilian.pilz@th-nuernberg.de> ([ORCID](#))
- Kevin Kunzmann <kevin.kunzmann@boehringer-ingenelheim.com> ([ORCID](#)) [copyright holder]
- Jan Meis <meis@imbi.uni-heidelberg.de> ([ORCID](#))
- Nico Bruder <bruder@imbi.uni-heidelberg.de> ([ORCID](#))

**See Also**

Useful links:

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

---

ANOVA-class

*Analysis of Variance*

---

**Description**

ANOVA is used to test whether there is a significant difference between the means of groups. The sample size which `adoptr` returns is the group wise sample size. The function `get_tau_ANOVA` is used to obtain a parameter  $\tau$ , which is used in the same way as  $\theta$  to describe the difference of means between the groups.

**Usage**

```
ANOVA(n_groups)
```

```
get_tau_ANOVA(means, common_sd = 1)
```

**Arguments**

<code>n_groups</code>	number of groups to be compared
<code>means</code>	vector denoting the mean per group
<code>common_sd</code>	standard deviation of the groups

**See Also**

see [probability\\_density\\_function](#) and [cumulative\\_distribution\\_function](#) to evaluate the pdf and the cdf, respectively. Use [NestedModels](#) to get insights in the implementation of ANOVA.

**Examples**

```
model <- ANOVA(3L)

H1 <- PointMassPrior(get_tau_ANOVA(c(0.4, 0.8, 0.5)), 1)
```

---

AverageN2-class	<i>Regularization via L1 norm</i>
-----------------	-----------------------------------

---

**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**

```
AverageN2(label = NA_character_)

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

**Arguments**

label	object label (string)
s	<a href="#">Score</a> 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 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  $X_1 = \sqrt{n}(r_E - r_C) / \sqrt{2r_0(1 - r_0)}$ , 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**

<code>rate_control</code>	assumed response rate in control group
<code>two_armed</code>	logical indicating if a two-armed trial is regarded
<code>x</code>	outcome
<code>probs</code>	vector of probabilities
<code>n</code>	sample size
<code>theta</code>	distribution parameter
<code>...</code>	further optional arguments
<code>object</code>	object of class <code>Binomial</code>
<code>nsim</code>	number of simulation runs
<code>seed</code>	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 'ContinuousPrior'  
bounds(dist, ...)  
  
## S4 method for signature 'PointMassPrior'  
bounds(dist, ...)
```

## Arguments

`dist` a univariate [distribution](#) object  
`...` further optional arguments

## Value

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

**Examples**

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

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

---

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**

<code>d</code>	design
<code>x1</code>	stage-one test statistic
<code>...</code>	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

```

---

ChiSquared-class

*Chi-Squared data distribution*


---

## Description

Implements a chi-squared distribution. The classes `Pearson2xk` and `ZSquared` are subclasses, used in two different situations. `Pearson2xK` is used when testing  $k$  groups for homogeneity in response rates. The null hypothesis is  $r_1 = \dots = r_k$ , and the alternative is that there exists a pair of groups with differing rates. `ZSquared` implements the square of a normally distributed random variable with mean  $\mu$  and standard deviation  $\sigma^2$ .

## Usage

```
ChiSquared(df)
```

```
## S4 method for signature 'ChiSquared'
```

```
quantile(x, probs, n, theta, ...)
```

```
## S4 method for signature 'ChiSquared,numeric'
```

```
simulate(object, nsim, n, theta, seed = NULL, ...)
```

## Arguments

<code>df</code>	number of degrees of freedom
<code>x</code>	outcome
<code>probs</code>	vector of probabilities
<code>n</code>	sample size

theta	distribution parameter
...	further optional arguments
object	object of class ChiSquared
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 <- ChiSquared(df=4)
```

---

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	<i>Condition a prior on an interval</i>
-----------	---

---

**Description**

Restrict an object of class [Prior](#) to a sub-interval and re-normalize the PDF.

**Usage**

```

condition(dist, interval, ...)

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

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

```

**Arguments**

dist	a univariate <a href="#">distribution</a> 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(
  ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4)),
  c(.3, .5)
)
bounds(tmp) # c(.3, .4)

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

---

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**

```
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 <b>distribution</b> object
prior	a <b>Prior</b> object
label	object label (string)
s	<b>Score</b> 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  $n(d, x_1)$  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)]")

ExpectedNumberOfEvents(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 <a href="#">distribution</a> object
prior	a <a href="#">Prior</a> object
s	<a href="#">Score</a> 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)
ene <- ExpectedNumberOfEvents(Survival(0.7), PointMassPrior(1.7, 1))

# 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  $\leq$  x', 'x  $\leq$  score' and 'score  $\geq$  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 <code>length(n2_pivots)</code> if <code>length(n2_pivots) == length(c2_pivots) &gt; 1</code> , otherwise <code>c2</code> and <code>n2</code> are taken to be constant in stage-two and replicated to match the number of pivots specified by <code>order</code>
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 'ChiSquared,numeric,numeric,numeric'
cumulative_distribution_function(dist, x, n, theta, ...)

## S4 method for signature 'NestedModels,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, ...)

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

**Arguments**

<code>dist</code>	a univariate <a href="#">distribution</a> object
<code>x</code>	outcome
<code>n</code>	sample size
<code>theta</code>	distribution parameter
<code>...</code>	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(Pearson2xK(3), 1, 30, get_tau_Pearson2xK(c(0.3,0.4,0.7,0.2)))
cumulative_distribution_function(ZSquared(TRUE), 1, 35, get_tau_ZSquared(0.4, 1))

cumulative_distribution_function(ANOVA(3), 1, 30, get_tau_ANOVA(c(0.3, 0.4, 0.7, 0.2)))

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

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

cumulative_distribution_function(Survival(0.6,TRUE),0.75,50,0.9)
```

---

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	<i>Expected value of a function</i>
-------------	-------------------------------------

---

**Description**

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

**Usage**

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

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

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

**Arguments**

dist	a univariate <a href="#">distribution</a> object
f	a univariate function, must be vectorized
...	further optional arguments

**Value**

numeric, expected value of  $f$  with respect to  $\text{dist}$

**Examples**

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

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

---

get\_initial\_design      *Initial design*

---

**Description**

The optimization method `minimize` requires an initial design for optimization. This function provides a variety of possibilities to hand-craft designs that fulfill type I error and type II error constraints which may be used as initial designs.

**Usage**

```
get_initial_design(
  theta,
  alpha,
  beta,
  type_design = c("two-stage", "group-sequential", "one-stage"),
  type_c2 = c("linear_decreasing", "constant"),
  type_n2 = c("optimal", "constant", "linear_decreasing", "linear_increasing"),
  dist = Normal(),
  cf,
  ce,
  info_ratio = 0.5,
  slope,
  weight = sqrt(info_ratio),
  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	maximal type II error rate
type_design	type of design
type_c2	either linear-decreasing c2-function according to inverse normal combination test or constant c2
type_n2	design of n2-function
dist	distribution of the test statistic
cf	first-stage futility boundary
ce	first-stage efficacy boundary. Note that specifying this boundary implies that the type I error constraint might not be fulfilled anymore
info_ratio	the ratio between first and second stage sample size
slope	slope of n2 function
weight	weight of first stage test statistics in inverse normal combination test
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. The first stage efficacy boundary and the  $c_2$  boundary are chosen as Pocock-boundaries, so either  $c_e = c_2$  if  $c_2$  is constant or  $c_e = c$ , where the null hypothesis is rejected if  $w_1 Z_1 + w_2 Z_2 > c$ . By specifying  $ce$ , it's clear that the boundaries are not Pocock-boundaries anymore, so the type I error constraint may not be fulfilled. IMPORTANT: When using the t-distribution or ANOVA, the design does probably not keep the type I and type II error, only approximate designs are returned.

### Value

An object of class `TwoStageDesign`.

### Examples

```
init <- get_initial_design(
  theta = 0.3,
  alpha = 0.025,
  beta = 0.2,
  type_design="two-stage",
  type_c2="linear_decreasing",
  type_n2="linear_increasing",
  dist=Normal(),
  cf=0.7,
  info_ratio=0.5,
  slope=23,
  weight = 1/sqrt(3)
)
```

---

```
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 <code>get_lower_boundary_design</code> , respectively, to $c1f + c1\_buffer$ and $c1e + c1\_buffer$ in <code>get_upper_boundary_design</code> . 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, ...)

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

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

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

### Arguments

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

event\_rate      probability that a subject in either group will eventually have an event, only needs to be specified for time-to-event endpoints.

**See Also**

[TwoStageDesign](#) for superclass and inherited methods

**Examples**

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

```
design_survival <- GroupSequentialDesign(25, 0, 2, 25, c(1, 1.5, 2.5), event_rate = 0.7)
```

```
TwoStageDesign(design)
```

```
TwoStageDesign(design_survival)
```

---

GroupSequentialDesignSurvival-class

*Group-sequential two-stage designs for time-to-event-endpoints*

---

**Description**

Group-sequential designs for time-to-event-endpoints are a subclass of both [TwoStageDesignSurvival](#) and [GroupSequentialDesign](#).

**See Also**

[TwoStageDesignSurvival-class](#) and [GroupSequentialDesign-class](#) for superclasses and inherited methods.

---

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 minimization 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	<a href="#">Score</a> 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),
  c2_decreasing = FALSE,
  check_constraints = TRUE,
```

```

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
c2_decreasing	if TRUE, the c2_pivots are forced to be monotonically decreasing
check_constraints	if TRUE, it is checked if constraints are fulfilled
opts	options list passed to nloptr
...	further optional arguments passed to <a href="#">nloptr</a>

### 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 'OneStageDesign,numeric'
n2(d, x1, ...)

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

```

### Arguments

<code>d</code>	design
<code>...</code>	further optional arguments
<code>round</code>	logical should sample sizes be rounded to next integer?
<code>x1</code>	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**

<code>label</code>	object label (string)
<code>s</code>	<a href="#">Score</a> object
<code>design</code>	object
<code>optimization</code>	logical, if TRUE uses a relaxation to real parameters of the underlying design; used for smooth optimization.
<code>...</code>	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
```

---

NestedModels-class      *F-Distribution*

---

**Description**

Implements the F-distribution used for an ANOVA or for the comparison of the fit of two nested regression models. In both cases, the test statistic follows a F-distribution. `NestedModel` is used to compare the fit of two regression models, where one model contains the independent variables of the smaller model as a subset. Then, one can use ANOVA to determine whether more variance can be explained by adding more independent variables. In the class `ANOVA`, the number of independent variables of the smaller model is set to 1 in order to match the degrees of freedom and we obtain a one-way ANOVA.

**Usage**

```
NestedModels(p_inner, p_outer)

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

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

**Arguments**

<code>p_inner</code>	number of independent variables in smaller model
<code>p_outer</code>	number of independent variables in bigger model
<code>x</code>	outcome
<code>probs</code>	vector of probabilities
<code>n</code>	sample size

theta	distribution parameter
...	further optional arguments
object	object of class NestedModels
nsim	number of simulation runs
seed	random seed

**Slots**

p_inner	number of parameters in smaller model
p_outer	number of parameters in bigger model

**See Also**

See [probability\\_density\\_function](#) and [cumulative\\_distribution\\_function](#) to evaluate the pdf and the cdf, respectively. Use [ANOVA](#) for detailed information of ANOVA.

**Examples**

```
model <- NestedModels(2, 4)
```

---

Normal-class	<i>Normal data distribution</i>
--------------	---------------------------------

---

**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  $n_2 = 0$ ,  $c = c_1^f = c_1^e$ ,  $c_2(x_1) = \text{ifelse}(x_1 < c, \text{Inf}, -\text{Inf})$ . No integration pivots etc are required (set to NaN).

**Usage**

```
OneStageDesign(n, ...)

## S4 method for signature 'numeric'
OneStageDesign(n, c, event_rate)

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

## S4 method for signature 'OneStageDesignSurvival'
```

```
TwoStageDesign(n1, order = 5L, eps = 0.01, ...)

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

### Arguments

n	sample size (stage-one sample size)
...	further optional arguments
c	rejection boundary ( $c = c_1^f = c_1^e$ )
event_rate	probability that a subject in either group will eventually have an event, only needs to be specified for time-to-event endpoints.
n1	OneStageDesign object to convert, overloaded from <a href="#">TwoStageDesign</a>
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 [c1f, c1e]; this is given by c +/- eps.
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-class](#)

### Examples

```
design <- OneStageDesign(30, 1.96)
summary(design)
design_twestage <- TwoStageDesign(design)
summary(design_twestage)
design_survival <- OneStageDesign(30, 1.96, 0.7)

TwoStageDesign(design_survival)
```

---

 OneStageDesignSurvival-class

*One-stage designs for time-to-event endpoints*


---

### Description

OneStageDesignSurvival is a subclass of both OneStageDesign and TwoStageDesignSurvival.

### See Also

[TwoStageDesignSurvival-class](#) and [OneStageDesign-class](#) for superclasses and inherited methods.

---

 Pearson2xK-class

*Pearson's chi-squared test for contingency tables*


---

### Description

When we test for homogeneity of rates in a  $k$ -armed trial with binary endpoints, the test statistic is chi-squared distributed with  $k - 1$  degrees of freedom under the null. Under the alternative, the statistic is chi-squared distributed with a non-centrality parameter  $\lambda$ . The function `get_tau_Pearson2xk` then computes  $\tau$ , such that  $\lambda$  is given as  $n \cdot \tau$ , where  $n$  is the number of subjects per group. In `adoptr`,  $\tau$  is used in the same way as  $\theta$  in the case of the normally distributed test statistic.

### Usage

```
Pearson2xK(n_groups)
```

```
get_tau_Pearson2xK(p_vector)
```

### Arguments

`n_groups`            number of groups considered for testing procedure

`p_vector`            vector denoting the event rates per group

### Examples

```
pearson <- Pearson2xK(3)
```

```
H1 <- PointMassPrior(get_tau_Pearson2xK(c(.3, .25, .4)), 1)
```

---

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 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](#).

## 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	<i>Compute posterior distribution</i>
-----------	---------------------------------------

---

## Description

Return posterior distribution given observing stage-one outcome.

## Usage

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

## Arguments

dist	a univariate <a href="#">distribution</a> object
prior	a <a href="#">Prior</a> object
x1	stage-one test statistic
n1	stage-one sample size
...	further optional arguments

## Value

Object of class [Prior](#)

## Examples

```
tmp <- ContinuousPrior(function(x) dunif(x, .2, .4), c(.2, .4))  
posterior(Normal(), tmp, 2, 20)  
  
posterior(Normal(), PointMassPrior(0, 1), 2, 20)
```

---

predictive_cdf	<i>Predictive CDF</i>
----------------	-----------------------

---

**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,ContinuousPrior,numeric'
predictive_cdf(
  dist,
  prior,
  x1,
  n1,
  k = 10 * (prior@support[2] - prior@support[1]) + 1,
  ...
)

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

**Arguments**

<code>dist</code>	a univariate <a href="#">distribution</a> object
<code>prior</code>	a <a href="#">Prior</a> object
<code>x1</code>	stage-one test statistic
<code>n1</code>	stage-one sample size
<code>...</code>	further optional arguments
<code>k</code>	number of pivots for crude integral approximation

**Value**

numeric, value of the predictive CDF

**Examples**

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

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

---

predictive_pdf	<i>Predictive PDF</i>
----------------	-----------------------

---

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

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

### Arguments

<code>dist</code>	a univariate <a href="#">distribution</a> object
<code>prior</code>	a <a href="#">Prior</a> object
<code>x1</code>	stage-one test statistic
<code>n1</code>	stage-one sample size
<code>...</code>	further optional arguments
<code>k</code>	number of pivots for crude integral approximation

### Value

numeric, value of the predictive PDF

### Examples

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

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

---

```
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	<i>Univariate prior on model parameter</i>
-------------	--

---

**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)
)
```

---

```
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 'ChiSquared,numeric,numeric,numeric'
probability_density_function(dist, x, n, theta, ...)

## S4 method for signature 'NestedModels,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, ...)

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

### Arguments

<code>dist</code>	a univariate <a href="#">distribution</a> object
<code>x</code>	outcome
<code>n</code>	sample size
<code>theta</code>	distribution parameter
<code>...</code>	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(Pearson2xK(3), 1, 30, get_tau_Pearson2xK(c(0.3, 0.4, 0.7, 0.2)))
probability_density_function(ZSquared(TRUE), 1, 35, get_tau_ZSquared(0.4, 1))

probability_density_function(ANOVA(3), 1, 30, get_tau_ANOVA(c(0.3, 0.4, 0.7, 0.2)))

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

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

probability_density_function(Survival(0.6, TRUE), 0.75, 50, 0.9)
```

---

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	<i>Create a collection of constraints</i>
------------	---

---

### 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  
)
```

---

Survival-class      *Log-rank test*

---

### Description

Implements the normal approximation of the log-rank test statistic.

### Usage

```
Survival(event_rate, two_armed = TRUE)

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

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

### Arguments

event_rate	probability that a subject will eventually have an event
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 Survival
nsim	number of simulation runs
seed	random seed

### Slots

event\_rate cf. parameter 'event\_rate'

### See Also

see [probability\\_density\\_function](#) and [cumulative\\_distribution\\_function](#) to evaluate the pdf and the cdf, respectively.

### Examples

```
datadist <- Survival(event_rate=0.6, two_armed=TRUE)
```

---

SurvivalDesign	<i>SurvivalDesign</i>
----------------	-----------------------

---

## Description

SurvivalDesign is a function that converts an arbitrary design to a survival design.

## Usage

```
SurvivalDesign(design, event_rate)

## S4 method for signature 'TwoStageDesign'
SurvivalDesign(design, event_rate)

## S4 method for signature 'TwoStageDesign'
TwoStageDesign(n1, event_rate)

## S4 method for signature 'OneStageDesign'
OneStageDesign(n, event_rate)

## S4 method for signature 'OneStageDesign'
SurvivalDesign(design, event_rate)

## S4 method for signature 'GroupSequentialDesign'
GroupSequentialDesign(n1, event_rate)

## S4 method for signature 'GroupSequentialDesign'
SurvivalDesign(design, event_rate)
```

## Arguments

design	design that should be converted to a survival design
event_rate	probability that a subject in either group will eventually have an event
n1	design object to convert (overloaded from TwoStageDesign)
n	design object to convert (overloaded from TwoStageDesign)

## Value

Converts any type of design to a survival design

## Examples

```
design <- get_initial_design(0.4, 0.025, 0.1)
SurvivalDesign(design, 0.8)

design_os <- get_initial_design(0.4, 0.025, 0.1, type_design = "one-stage")
design_gs <- get_initial_design(0.4, 0.025, 0.1, type_design = "group-sequential")
```

```
OneStageDesign(design_os, 0.7)
GroupSequentialDesign(design_gs, 0.8)
```

---

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 <code>tunable_parameters</code>

### 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  $(n_1, c_1^f, c_1^e, n_2(\cdot), c_2(\cdot))$ . 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,
  event_rate,
  ...
)

## 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 <code>length(n2_pivots)</code> if <code>length(n2_pivots) == length(c2_pivots) &gt; 1</code> , otherwise c2 and n2 are taken to be constant in stage-two and replicated to match the number of pivots specified by order
event_rate	probability that a subject in either group will eventually have an event, only needs to be specified for time-to-event endpoints
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 <code>adoptr:::scaled_integration_pivots(design)</code>
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 <code>minimize</code> or not; cf. <code>make_fixed</code> )

### 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)
```

---

TwoStageDesignSurvival-class

*Two-stage design for time-to-event-endpoints*

---

### Description

When conducting a study with time-to-event endpoints, the main interest is not the sample size, but the number of overall necessary events. Thus, `adoptr` does not use the sample size for calculating the design. Instead, it uses the number of events directly. In the framework of `adoptr`, all the calculations are done group-wise, where both of the groups are equal-sized. This means, that the number of events `adoptr` has computed is only half of the overall number of necessary events. In order to facilitate this issue, the look of the summary and show functions have been changed in the survival analysis setting. The sample size is implicitly determined by dividing the number of events by the event rate. Survival objects are only created, when the argument `event_rate` is not missing.

### Slots

`event_rate` probability that a subject in either group will eventually have an event

### See Also

`TwoStageDesign` for superclass and inherited methods

---

`ZSquared-class`*Distribution class of a squared normal distribution*

---

**Description**

Implementation of  $Z^2$ , where  $Z$  is normally distributed with mean  $\mu$  and variance  $\sigma^2$ .  $Z^2$  is chi-squared distributed with 1 degree of freedom and non-centrality parameter  $(\mu/\sigma)^2$ . The function `get_tau_ZSquared` computes the factor  $\tau = (\mu/\sigma)^2$ , such that  $\tau$  is the equivalent of  $\theta$  in the normally distributed case. The square of a normal distribution  $Z^2$  can be used for two-sided hypothesis testing.

**Usage**

```
ZSquared(two_armed = TRUE)

get_tau_ZSquared(mu, sigma)
```

**Arguments**

<code>two_armed</code>	logical indicating if a two-armed trial is regarded
<code>mu</code>	mean of $Z$
<code>sigma</code>	standard deviation of $Z$

**Examples**

```
zsquared <- ZSquared(FALSE)

H1 <- PointMassPrior(get_tau_ZSquared(0.4, 1), 1)
```

# Index

$\leq$ , ConditionalScore, ConditionalScore-method (Constraints), 14  
 $\leq$ , ConditionalScore, numeric-method (Constraints), 14  
 $\leq$ , UnconditionalScore, UnconditionalScore-method (Constraints), 14  
 $\leq$ , UnconditionalScore, numeric-method (Constraints), 14  
 $\leq$ , numeric, ConditionalScore-method (Constraints), 14  
 $\leq$ , numeric, UnconditionalScore-method (Constraints), 14  
 $\geq$ , ConditionalScore, ConditionalScore-method (Constraints), 14  
 $\geq$ , ConditionalScore, numeric-method (Constraints), 14  
 $\geq$ , UnconditionalScore, UnconditionalScore-method (Constraints), 14  
 $\geq$ , UnconditionalScore, numeric-method (Constraints), 14  
 $\geq$ , numeric, ConditionalScore-method (Constraints), 14  
 $\geq$ , numeric, UnconditionalScore-method (Constraints), 14  
  
 adoptr, 3, 51, 53  
 adoptr-package (adoptr), 3  
 ANOVA, 3, 32  
 ANOVA (ANOVA-class), 4  
 ANOVA-class, 4  
 AverageN2, 5, 31  
 AverageN2 (AverageN2-class), 5  
 AverageN2-class, 5  
  
 Binomial, 3, 18, 42  
 Binomial (Binomial-class), 6  
 Binomial-class, 6  
 bounds, 7, 41  
 bounds, ContinuousPrior-method (bounds), 7  
 bounds, PointMassPrior-method (bounds), 7  
 c2, 8, 30, 53  
 c2, OneStageDesign, numeric-method (c2), 8  
 c2, TwoStageDesign, numeric-method (c2), 8  
 ChiSquared, 3  
 ChiSquared (ChiSquared-class), 9  
 ChiSquared-class, 9  
 composite, 10, 44  
 condition, 11, 41  
 condition, ContinuousPrior, numeric-method (condition), 11  
 condition, PointMassPrior, numeric-method (condition), 11  
 ConditionalPower, 3, 44  
 ConditionalPower (ConditionalPower-class), 12  
 ConditionalPower-class, 12  
 ConditionalSampleSize, 3, 27, 44  
 ConditionalSampleSize (ConditionalSampleSize-class), 13  
 ConditionalSampleSize-class, 13  
 ConditionalScore, 36  
 ConstraintCollection (subject\_to), 47  
 Constraints, 14  
 ContinuousPrior, 3, 37, 41  
 ContinuousPrior (ContinuousPrior-class), 16  
 ContinuousPrior-class, 16  
 cumulative\_distribution\_function, 4, 7, 10, 17, 32, 33, 46, 48  
 cumulative\_distribution\_function, Binomial, numeric, numeric, (cumulative\_distribution\_function), 17  
 cumulative\_distribution\_function, ChiSquared, numeric, numeric, (cumulative\_distribution\_function), 17  
 cumulative\_distribution\_function, NestedModels, numeric, numeric, (cumulative\_distribution\_function),

- [17](#)  
 cumulative\_distribution\_function, Normal, numeric, numeric-method  
 (cumulative\_distribution\_function), [13](#)  
[17](#)  
 cumulative\_distribution\_function, Student, numeric, numeric-method  
 (cumulative\_distribution\_function), [20](#)  
[17](#)  
 cumulative\_distribution\_function, Survival, numeric, numeric-method  
 (cumulative\_distribution\_function), [22](#)  
[17](#)  
 DataDistribution, [3](#), [39–41](#), [43](#)  
 DataDistribution  
 (DataDistribution-class), [18](#)  
 DataDistribution-class, [18](#)  
 distribution, [7](#), [11–13](#), [18](#), [19](#), [38–40](#), [42](#)  
 evaluate (Scores), [43](#)  
 evaluate, AverageN2, TwoStageDesign-method  
 (AverageN2-class), [5](#)  
 evaluate, CompositeScore, TwoStageDesign-method  
 (composite), [10](#)  
 evaluate, ConditionalPower, TwoStageDesign-method  
 (ConditionalPower-class), [12](#)  
 evaluate, ConditionalSampleSize, TwoStageDesign-method  
 (ConditionalSampleSize-class), [13](#)  
[13](#)  
 evaluate, Constraint, TwoStageDesign-method  
 (Constraints), [14](#)  
 evaluate, ConstraintsCollection, TwoStageDesign-method  
 (subject\_to), [47](#)  
 evaluate, IntegralScore, TwoStageDesign-method  
 (Scores), [43](#)  
 evaluate, MaximumSampleSize, TwoStageDesign-method  
 (MaximumSampleSize-class), [26](#)  
 evaluate, N1, TwoStageDesign-method  
 (N1-class), [30](#)  
 expectation, [19](#), [41](#)  
 expectation, ContinuousPrior, function-method  
 (expectation), [19](#)  
 expectation, PointMassPrior, function-method  
 (expectation), [19](#)  
 expected (Scores), [43](#)  
 expected, ConditionalScore-method  
 (Scores), [43](#)  
 ExpectedNumberOfEvents  
 (ConditionalSampleSize-class),  
[13](#)  
 ExpectedSampleSize, [3](#)  
 ExpectedSampleSize  
 (ExpectedSampleSize-class),  
[13](#)  
 get\_initial\_design, numeric-method  
 (get\_initial\_design), [20](#)  
 get\_lower\_boundary\_design, [22](#)  
 get\_lower\_boundary\_design, GroupSequentialDesign-method  
 (get\_lower\_boundary\_design), [22](#)  
 get\_lower\_boundary\_design, OneStageDesign-method  
 (get\_lower\_boundary\_design), [22](#)  
 get\_lower\_boundary\_design, TwoStageDesign-method  
 (get\_lower\_boundary\_design), [22](#)  
 get\_tau\_ANOVA (ANOVA-class), [4](#)  
 get\_tau\_Pearson2xK (Pearson2xK-class),  
[35](#)  
 get\_tau\_ZSquared (ZSquared-class), [54](#)  
 get\_upper\_boundary\_design  
 (get\_lower\_boundary\_design), [22](#)  
 get\_upper\_boundary\_design, GroupSequentialDesign-method  
 (get\_lower\_boundary\_design), [22](#)  
 get\_upper\_boundary\_design, OneStageDesign-method  
 (get\_lower\_boundary\_design), [22](#)  
 get\_upper\_boundary\_design, TwoStageDesign-method  
 (get\_lower\_boundary\_design), [22](#)  
 GroupSequentialDesign, [3](#)  
 GroupSequentialDesign  
 (GroupSequentialDesign-class),  
[24](#)  
 GroupSequentialDesign, GroupSequentialDesign-method  
 (SurvivalDesign), [49](#)  
 GroupSequentialDesign, numeric-method  
 (GroupSequentialDesign-class),  
[24](#)  
 GroupSequentialDesign-class, [24](#)  
 GroupSequentialDesignSurvival-class,  
[25](#)  
 make\_fixed, [50](#), [52](#)  
 make\_fixed (make\_tunable), [25](#)  
 make\_fixed, TwoStageDesign-method  
 (make\_tunable), [25](#)  
 make\_tunable, [25](#), [50](#), [53](#)  
 make\_tunable, TwoStageDesign-method  
 (make\_tunable), [25](#)  
 MaximumSampleSize  
 (MaximumSampleSize-class), [26](#)  
 MaximumSampleSize-class, [26](#)  
 minimize, [15](#), [20](#), [22](#), [26](#), [27](#), [29](#), [30](#), [47](#), [52](#)

- n, [8](#), [53](#)
- n(n1), [29](#)
- n,TwoStageDesign,numeric-method (n1), [29](#)
- N1, [5](#), [31](#)
- N1 (N1-class), [30](#)
- n1, [29](#)
- n1,TwoStageDesign-method (n1), [29](#)
- N1-class, [30](#)
- n2(n1), [29](#)
- n2,GroupSequentialDesign,numeric-method (n1), [29](#)
- n2,OneStageDesign,numeric-method (n1), [29](#)
- n2,TwoStageDesign,numeric-method (n1), [29](#)
- NestedModels, [4](#)
- NestedModels (NestedModels-class), [31](#)
- NestedModels-class, [31](#)
- nloptr, [28](#)
- Normal, [3](#), [18](#), [42](#)
- Normal (Normal-class), [32](#)
- Normal-class, [32](#)
- one-stage designs, [53](#)
- OneStageDesign, [3](#)
- OneStageDesign (OneStageDesign-class), [33](#)
- OneStageDesign,numeric-method (OneStageDesign-class), [33](#)
- OneStageDesign,OneStageDesign-method (SurvivalDesign), [49](#)
- OneStageDesign-class, [33](#)
- OneStageDesignSurvival-class, [35](#)
- Pearson2xK, [3](#)
- Pearson2xK (Pearson2xK-class), [35](#)
- Pearson2xK-class, [35](#)
- plot,OneStageDesign-method (OneStageDesign-class), [33](#)
- plot,TwoStageDesign-method, [36](#)
- PointMassPrior, [3](#), [17](#), [41](#)
- PointMassPrior (PointMassPrior-class), [37](#)
- PointMassPrior-class, [37](#)
- posterior, [38](#), [41](#)
- posterior,DataDistribution,ContinuousPrior,numeric-method (posterior), [38](#)
- posterior,DataDistribution,PointMassPrior,numeric-method (posterior), [38](#)
- Power, [3](#)
- Power (ConditionalPower-class), [12](#)
- predictive\_cdf, [39](#), [41](#)
- predictive\_cdf,DataDistribution,ContinuousPrior,numeric-method (predictive\_cdf), [39](#)
- predictive\_cdf,DataDistribution,PointMassPrior,numeric-method (predictive\_cdf), [39](#)
- predictive\_pdf, [40](#), [41](#)
- predictive\_pdf,DataDistribution,ContinuousPrior,numeric-method (predictive\_pdf), [40](#)
- predictive\_pdf,DataDistribution,PointMassPrior,numeric-method (predictive\_pdf), [40](#)
- print, [52](#)
- print (print.adoptrOptimizationResult), [41](#)
- print.adoptrOptimizationResult, [41](#)
- Prior, [11–13](#), [16](#), [37–40](#), [43](#)
- Prior (Prior-class), [41](#)
- Prior-class, [41](#)
- probability\_density\_function, [4](#), [7](#), [10](#), [32](#), [33](#), [42](#), [46](#), [48](#)
- probability\_density\_function,Binomial,numeric,numeric,numeric-method (probability\_density\_function), [42](#)
- probability\_density\_function,ChiSquared,numeric,numeric,numeric-method (probability\_density\_function), [42](#)
- probability\_density\_function,NestedModels,numeric,numeric,numeric-method (probability\_density\_function), [42](#)
- probability\_density\_function,Normal,numeric,numeric,numeric-method (probability\_density\_function), [42](#)
- probability\_density\_function,Student,numeric,numeric,numeric-method (probability\_density\_function), [42](#)
- probability\_density\_function,Survival,numeric,numeric,numeric-method (probability\_density\_function), [42](#)
- quantile,Binomial-method (Binomial-class), [6](#)
- quantile,ChiSquared-method (ChiSquared-class), [9](#)
- quantile,NestedModels-method (NestedModels-class), [31](#)
- quantile,Normal-method (Normal-class), [32](#)

- quantile, Student-method  
(Student-class), 46
- quantile, Survival-method  
(Survival-class), 48
- Score, 5, 12, 13, 15, 27, 30, 43
- Scores, 3, 11, 13, 14, 27, 43
- simulate, 53
- simulate, Binomial, numeric-method  
(Binomial-class), 6
- simulate, ChiSquared, numeric-method  
(ChiSquared-class), 9
- simulate, NestedModels, numeric-method  
(NestedModels-class), 31
- simulate, Normal, numeric-method  
(Normal-class), 32
- simulate, Student, numeric-method  
(Student-class), 46
- simulate, Survival, numeric-method  
(Survival-class), 48
- simulate, TwoStageDesign, numeric-method,  
45
- Student, 3
- Student (Student-class), 46
- Student-class, 46
- subject\_to, 47
- summary, TwoStageDesign-method  
(TwoStageDesign-class), 51
- Survival, 3
- Survival (Survival-class), 48
- Survival-class, 48
- SurvivalDesign, 49
- SurvivalDesign, GroupSequentialDesign-method  
(SurvivalDesign), 49
- SurvivalDesign, OneStageDesign-method  
(SurvivalDesign), 49
- SurvivalDesign, TwoStageDesign-method  
(SurvivalDesign), 49
- tunable\_parameters, 26, 50, 53
- tunable\_parameters, TwoStageDesign-method  
(tunable\_parameters), 50
- TwoStageDesign, 3, 8, 21, 23–26, 29, 30, 33,  
34, 36, 45, 50, 51, 53
- TwoStageDesign (TwoStageDesign-class),  
51
- TwoStageDesign, GroupSequentialDesign-method  
(GroupSequentialDesign-class),  
24
- TwoStageDesign, GroupSequentialDesignSurvival-method  
(GroupSequentialDesign-class),  
24
- TwoStageDesign, numeric-method  
(TwoStageDesign-class), 51
- TwoStageDesign, OneStageDesign-method  
(OneStageDesign-class), 33
- TwoStageDesign, OneStageDesignSurvival-method  
(OneStageDesign-class), 33
- TwoStageDesign, TwoStageDesign-method  
(SurvivalDesign), 49
- TwoStageDesign-class, 51
- TwoStageDesignSurvival-class, 53
- UnconditionalScore, 52
- update, OneStageDesign-method  
(tunable\_parameters), 50
- update, TwoStageDesign-method  
(tunable\_parameters), 50
- ZSquared, 3
- ZSquared (ZSquared-class), 54
- ZSquared-class, 54