Package ‘adestr’

July 12, 2024

Type Package

Title Estimation in Optimal Adaptive Two-Stage Designs

Version 1.0.0

Description Methods to evaluate the performance characteristics of various point and interval estimators for optimal adaptive two-stage designs as described in Meis et al. (2024) <doi:10.1002/sim.10020>. Specifically, this package is written to work with trial designs created by the 'adoptr' package (Kunzmann et al. (2021) <doi:10.18637/jss.v098.i09>; Pilz et al. (2021) <doi:10.1002/sim.8953>)). Apart from the a priori evaluation of performance characteristics, this package also allows for the evaluation of the implemented estimators on real datasets, and it implements methods to calculate p-values.

License GPL (>= 2)

Copyright This package contains a modified version of the monotonic spline functions from the 'stats' package. Specifically, the code is containted in the files 'R/fastmonoHFC.R', 'src/fastmonoHFC.c', 'src/modreg.h' and 'src/monoSpl.c'. The R Core team and Martin Maechler are the copyright holders of the original code. Jan Meis is the copyright holder of everything else.

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.2.3

Depends R (>= 4.0.0), adoptr

Imports methods, stats, grDevices, cubature, ggplot2, ggpubr, scales, latex2exp, forcats, future.apply, progressr, Rdpack

Suggests covr, knitr, rmarkdown, testthat (>= 3.0.0), microbenchmark

Config/testthat/edition 3

Collate 'adestr_package.R' 'twostagedesign_with_cache.R' 'analyze.R' 'estimators.R' 'densities.R' 'evaluate_estimator.R' 'fastmonoHFC.R' 'fisher_information.R' 'hcubature.R' 'helper_functions.R' 'integrate_over_sample_space.R' 'reference_implementation.R' 'mle_distribution.R' 'mlmse_score.R' 'n2c2_helpers.R' 'plot.R' 'priors.R' 'print.R'

URL https://jan-imbi.github.io/adestr/

RdMacros Rdpack

NeedsCompilation yes

Author Jan Meis [aut, cre] (<https://orcid.org/0000-0001-5407-7220>),
Martin Maechler [cph] (<https://orcid.org/0000-0002-8685-9910>,
Original author of monoSpl.c (from the 'stats' package).)

Maintainer Jan Meis <meis@imbi.uni-heidelberg.de>

Repository CRAN

Date/Publication 2024-07-12 13:50:09 UTC

Contents

`adestr` *adestr*

Description

Point estimates, confidence intervals, and p-values for optimal adaptive two-stage designs.

Details

This package implements methods to evaluate the performance characteristics of various point and interval estimators for optimal adaptive two-stage designs. Specifically, this package is written to interface with trial designs created by the `adoptr` package (Kunzmann et al. 2021; Pilz et al. 2021). Apart from the a priori evaluation of performance characteristics, this package also allows for the calculation of the values of the estimators given real datasets, and it implements methods to calculate p-values.

Author(s)

Maintainer: Jan Meis `<meis@imbi.uni-heidelberg.de>` (ORCID)

Other contributors:

- Martin Maechler `<maechler@stat.math.ethz.ch>` (ORCID) (Original author of monoSpl.c (from the 'stats' package).) [copyright holder]

References

Kunzmann K, Pilz M, Herrmann C, Rauch G, Kieser M (2021). "The adoptr package: Adaptive Optimal Designs for Clinical Trials in R." *Journal of Statistical Software*, **98**(9), 1–21. doi:10.18637/jss.v098.i09.

Pilz M, Kunzmann K, Herrmann C, Rauch G, Kieser M (2021). "Optimal planning of adaptive two-stage designs." *Statistics in Medicine*, **40**(13), 3196-3213. doi:10.1002/sim.8953.

See Also

`evaluate_estimator`

`analyze`

`Statistic` `PointEstimator` `IntervalEstimator` `PValue`

`plot` `plot_p`

`https://jan-imbi.github.io/adestr/`

analyze *Analyze a dataset*

Description

The analyze function can be used calculate the values of a list of point estimators, confidence intervals, and p-values for a given dataset.

Usage

```
analyze(
  data,
  statistics = list(),
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'data.frame'
analyze(
  data,
  statistics = list(),
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)
```

Arguments

`data`	a data.frame containing the data to be analyzed.
`statistics`	a list of objects of class `PointEstimator`, `ConfidenceInterval` or `PValue`.
`data_distribution`	object of class `Normal` or `Student`.
`use_full_twoarm_sampling_distribution`	logical indicating whether this estimator is intended to be used with the full sampling distribution in a two-armed trial.
`design`	object of class `TwoStageDesign`.
`sigma`	assumed standard deviation.
`exact`	logical indicating usage of exact n2 function.

Details

Note that in adestr, statistics are codes as functions of the stage-wise sample means (and stage-wise sample variances if data_distribution is Student). In a first-step, the data is summarized to produce these parameters. Then, the list of statistics are evaluated at the values of these parameters.

The output of the `analyze` function also displays information on the hypothesis test and the interim decision. If the statistics list is empty, this will be the only information displayed.

Value

`Results` object containing the values of the statistics when applied to data.

Examples

```
set.seed(123)
dat <- data.frame(
  endpoint = c(rnorm(28, 0.3)),
  stage = rep(1, 28)
)
analyze(data = dat,
        statistics = list(),
        data_distribution = Normal(FALSE),
        design = get_example_design(),
        sigma = 1)

# The results suggest recruiting 32 patients for the second stage
dat <- rbind(
  dat,
  data.frame(
    endpoint = rnorm(32, mean = 0.3),
    stage = rep(2, 32)))
analyze(data = dat,
        statistics = get_example_statistics(),
        data_distribution = Normal(FALSE),
        design = get_example_design(),
        sigma = 1)
```

`c,EstimatorScoreResult-method`

Combine EstimatoreScoreResult objects into a list

Description

Creates an object of class EstimatoreScoreResultList, which is a basically list with the respective EstimatoreScoreResult objects.

Usage

```
## S4 method for signature 'EstimatorScoreResult'
c(x, ...)
```

Arguments

`x`	an object of class EstimatorScoreResult.
`...`	additional arguments passed along to the `list` function

Value

an object of class EstimatoreScoreResultList.

`c,EstimatorScoreResultList-method` *Combine EstimatoreScoreResult objects into a list*

Description

Creates an object of class EstimatoreScoreResultList, which is a basically list with the respective EstimatoreScoreResult objects.

Usage

```
## S4 method for signature 'EstimatorScoreResultList'
c(x, ...)
```

Arguments

`x`	an object of class EstimatorScoreResult.
`...`	additional arguments passed along to the `list` function

Value

an object of class EstimatoreScoreResultList.

`c2_extrapol` *Calculate the second-stage critical value for a design with cached spline parameters*

Description

Also extrapolates results for values outside of [c1f, c1e].

Usage

```
c2_extrapol(design, x1)
```

Arguments

`design`	an object of class `TwoStageDesignWithCache`.
`x1`	first-stage test statistic

`EstimatorScore-class` *Performance scores for point and interval estimators*

Description

These classes encode various metrics which can be used to evaluate the performance characteristics of point and interval estimators.

Usage

```
Expectation()

Bias()

Variance()

MSE()

OverestimationProbability()

Coverage()

SoftCoverage(shrinkage = 1)

Width()

TestAgreement()

Centrality(interval = NULL)
```

Arguments

`shrinkage`	shrinkage factor for bump function.
`interval`	confidence interval with respect to which centrality of a point estimator should be evaluated.

Value

an object of class `EstimatorScore`. This class signals that an object can be used with the `evaluate_estimator` function.

Slots

`label` name of the performance score. Used in printing methods.

Details on the implemented estimators

In the following, precise definitions of the performance scores implemented in adestr are given. To this end, let $\hat{\mu}$ denote a point estimator, $(\hat{l}, \hat{u})$ an interval estimator, denote the expected value of a random variable by $\mathbb{E}$, the probability of an event by P, and let μ be the real value of the underlying parameter to be estimated.

Scores for point estimators (`PointEstimatorScore`)::

- `Expectation()`: $\mathbb{E}[\hat{\mu}]$
- `Bias()`: $\mathbb{E}[\hat{\mu} - \mu]$
- `Variance()`: $\mathbb{E}[(\hat{\mu} - \mathbb{E}[\hat{\mu}])^2]$
- `MSE()`: $\mathbb{E}[(\hat{\mu} - mu)^2]$
- `OverestimationProbability()`: $P(\hat{\mu} > \mu)$
- `Centrality(interval)`: $\mathbb{E}[(\hat{\mu} - \hat{l}) + (\hat{\mu} - \hat{u}]$

Scores for confidence intervals (`IntervalEstimatorScore`)::

- `Coverage()`: $P(\hat{l} \leq \mu \leq \hat{u})$
- `Width()`: $\mathbb{E}[\hat{u} - \hat{l}]$
- `TestAgreement()`: $P\left(\left(\{0 < \hat{l} \text{ and } (c_{1,e} < Z_1 \text{ or } c_2(Z_1) < Z_2)\right) \text{ or } \left(\{\hat{l} \leq 0 \text{ and } (Z_1 < c_{1,f} \text{ or } Z_2 \leq c_2(Z_1))\}\right)\right)$

See Also

evaluate_estimator

Examples

```
evaluate_estimator(
  score = MSE(),
  estimator = SampleMean(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu = c(0, 0.3, 0.6),
  sigma = 1,
  exact = FALSE
)

evaluate_estimator(
  score = Coverage(),
  estimator = StagewiseCombinationFunctionOrderingCI(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu = c(0, 0.3),
  sigma = 1,
  exact = FALSE
)
```

evaluate_estimator *Evaluate performance characteristics of an estimator*

Description

This function evaluates an EstimatorScore for a PointEstimator or and IntervalEstimator by integrating over the sampling distribution.

Usage

```
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
 tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)
```

Arguments

score	performance measure to evaluate.
estimator	object of class PointEstimator, IntervalEstimator or PValue.
data_distribution	object of class Normal or Student.
use_full_twoarm_sampling_distribution	logical indicating whether this estimator is intended to be used with the full sampling distribution in a two-armed trial.
design	object of class TwoStageDesign.
true_parameter	true value of the parameter (used e.g. when evaluating bias).
mu	expected value of the underlying normal distribution.
sigma	assumed standard deviation.
tol	relative tolerance.

maxEval	maximum number of iterations.
absError	absolute tolerance.
exact	logical indicating usage of exact n2 function.
early_futility_part	include early futility part of integral.
continuation_part	include continuation part of integral.
early_efficacy_part	include early efficacy part of integral.
conditional_integral	treat integral as a conditional integral.

Details

General:

First, a functional representation of the integrand is created by combining information from the EstimatorScore object (score) and the PointEstimator or IntervalEstimator object (estimator). The sampling distribution of a design is determined by the TwoStageDesign object (design) and the DataDistribution object (data_distribution), as well as the assumed parameters μ (mu) and σ (sigma). The other parameters control various details of the integration problem.

Other parameters:

For a two-armed data_distribution, if use_full_twoarm_sampling_distribution is TRUE, the sample means for both groups are integrated independently. If use_full_twoarm_sampling_distribution is FALSE, only the difference in sample means is integrated.

true_parameter controls which parameters is supposed to be estimated. This is usually mu, but could be set to sigma if one is interested in estimating the standard deviation.

If the parameter exact is set to FALSE (the default), the continuous version of the second-stage sample-size function n2 is used. Otherwise, an integer valued version of that function will be used, though this is considerably slower.

The parameters early_futility_part, continuation_part and early_efficacy_part control which parts of the sample-space should be integrated over (all default to TRUE). They can be used in conjunction with the parameter conditional_integral, which enables the calculation of the expected value of performance score conditional on reaching any of the selected integration regions.

Lastly, the paramters tol, maxEval, and absError control the integration accuracy. They are handed down to the hcubature function.

Value

an object of class EstimatorScoreResult containing the values of the evaluated EstimatorScore and information about the setting for which they were calculated (e.g. the estimator, data_distribution, design, mu, and sigma).

See Also

EstimatorScore

PointEstimator IntervalEstimator

plot

Examples

```
evaluate_estimator(
  score = MSE(),
  estimator = SampleMean(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu = c(0, 0.3, 0.6),
  sigma = 1,
  exact = FALSE
)

evaluate_estimator(
  score = Coverage(),
  estimator = StagewiseCombinationFunctionOrderingCI(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu = c(0, 0.3),
  sigma = 1,
  exact = FALSE
)
```

`evaluate_estimator-methods`
Evaluate performance characteristics of an estimator

Description

This function evaluates an `EstimatorScore` for a `PointEstimator` or and `IntervalEstimator` by integrating over the sampling distribution.

Usage

```
## S4 method for signature 'PointEstimatorScore,IntervalEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'IntervalEstimatorScore,PointEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'list,Estimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'Expectation,PointEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'Bias,PointEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'Variance,PointEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'MSE,PointEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
```

```
)

## S4 method for signature 'OverestimationProbability,PointEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'Coverage,IntervalEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'SoftCoverage,IntervalEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'Width,IntervalEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'TestAgreement,IntervalEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
 tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'TestAgreement,PValue'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
 tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)

## S4 method for signature 'Centrality,PointEstimator'
evaluate_estimator(
  score,
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  true_parameter = mu,
  mu,
  sigma,
  tol = getOption("adestr_tol_outer", default = .adestr_options[["adestr_tol_outer"]]),
  maxEval = getOption("adestr_maxEval_outer", default =
    .adestr_options[["adestr_maxEval_outer"]]),
  absError = getOption("adestr_absError_outer", default =
    .adestr_options[["adestr_absError_outer"]]),
  exact = FALSE,
  early_futility_part = TRUE,
  continuation_part = TRUE,
  early_efficacy_part = TRUE,
  conditional_integral = FALSE
)
```

Arguments

`score`	performance measure to evaluate.
`estimator`	object of class `PointEstimator`, `IntervalEstimator` or `PValue`.
`data_distribution`	object of class `Normal` or `Student`.
`use_full_twoarm_sampling_distribution`	logical indicating whether this estimator is intended to be used with the full sampling distribution in a two-armed trial.
`design`	object of class `TwoStageDesign`.
`true_parameter`	true value of the parameter (used e.g. when evaluating bias).
`mu`	expected value of the underlying normal distribution.
`sigma`	assumed standard deviation.
`tol`	relative tolerance.
`maxEval`	maximum number of iterations.
`absError`	absolute tolerance.
`exact`	logical indicating usage of exact n2 function.
`early_futility_part`	include early futility part of integral.
`continuation_part`	include continuation part of integral.
`early_efficacy_part`	include early efficacy part of integral.
`conditional_integral`	treat integral as a conditional integral.

Details

General:

First, a functional representation of the integrand is created by combining information from the `EstimatorScore` object (`score`) and the `PointEstimator` or `IntervalEstimator` object (`estimator`). The sampling distribution of a design is determined by the `TwoStageDesign` object (`design`) and the `DataDistribution` object (`data_distribution`), as well as the assumed parameters μ (mu) and σ (sigma). The other parameters control various details of the integration problem.

Other parameters:

For a two-armed `data_distribution`, if `use_full_twoarm_sampling_distribution` is `TRUE`, the sample means for both groups are integrated independently. If `use_full_twoarm_sampling_distribution` is `FALSE`, only the difference in sample means is integrated.

`true_parameter` controls which parameters is supposed to be estimated. This is usually `mu`, but could be set to `sigma` if one is interested in estimating the standard deviation.

If the parameter `exact` is set to `FALSE` (the default), the continuous version of the second-stage sample-size function `n2` is used. Otherwise, an integer valued version of that function will be used, though this is considerably slower.

The parameters `early_futility_part`, `continuation_part` and `early_efficacy_part` control which parts of the sample-space should be integrated over (all default to `TRUE`). They can be used in conjunction with the parameter `conditional_integral`, which enables the calculation of the expected value of performance score conditional on reaching any of the selected integration regions.

Lastly, the paramters `tol`, `maxEval`, and `absError` control the integration accuracy. They are handed down to the `hcubature` function.

Value

an object of class `EstimatorScoreResult` containing the values of the evaluated `EstimatorScore` and information about the setting for which they were calculated (e.g. the `estimator`, `data_distribution`, `design`, `mu`, and `sigma`).

See Also

EstimatorScore

PointEstimator IntervalEstimator

plot

Examples

```
evaluate_estimator(
  score = MSE(),
  estimator = SampleMean(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu = c(0, 0.3, 0.6),
  sigma = 1,
  exact = FALSE
)

evaluate_estimator(
  score = Coverage(),
  estimator = StagewiseCombinationFunctionOrderingCI(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu = c(0, 0.3),
  sigma = 1,
  exact = FALSE
)
```

---

evaluate_scenarios_parallel
Evaluate different scenarios in parallel

Description

This function takes a list of lists of scores, a list of lists of estimators, and lists lists of various other design parameters. Each possible combination of the elements of the respective sublists is then used to create separate scenarios. These scenarios are than evaluated independelty of each other, allowing for parallelization via the `future` framework. For each scenario, one call to the `evaluate_estimator` function is made.

Usage

```
evaluate_scenarios_parallel(
  score_lists,
  estimator_lists,
  data_distribution_lists,
  use_full_twoarm_sampling_distribution_lists,
  design_lists,
  true_parameter_lists,
  mu_lists,
  sigma_lists,
  tol_lists,
  maxEval_lists,
  absError_lists,
  exact_lists,
  early_futility_part_lists,
  continuation_part_lists,
  early_efficacy_part_lists,
  conditional_integral_lists
)
```

Arguments

`score_lists`	a list of lists of estimator scores.
`estimator_lists`	a list of lists of estimators.
`data_distribution_lists`	a list of lists of data distributions.
`use_full_twoarm_sampling_distribution_lists`	a list of lists of use_full_twoarm_sampling_distribution_lists parameters.

`design_lists`	a list of lists of designs.
`true_parameter_lists`	a list of lists of true parameters.
`mu_lists`	a list of lists of mu vectors.
`sigma_lists`	a list of lists of sigma values.
`tol_lists`	a list of lists of relative tolerances.
`maxEval_lists`	a list of lists of maxEval boundaries.
`absError_lists`	a list of lists of absError boundaries.
`exact_lists`	a list of lists of ‘exact‘ parameters.
`early_futility_part_lists`	a list of lists of ‘early_futility_part_lists‘ parameters.
`continuation_part_lists`	a list of lists of ‘continuation_part_lists‘ parameters.
`early_efficacy_part_lists`	a list of lists of ‘early_efficacy_part_lists‘ parameters.
`conditional_integral_lists`	a list of lists of ‘conditional_integral_lists‘ parameters.

Details

Concretely, the cross product of the first sublist of scores and the first sublist of estimators and the other parameters is calculated. Then the cross product of the second sublist of scores, estimators and other design parameters is calculated. All of these cross products together make up the set of all scenarios. The combinations say the first sublist of scores and the second sublist of estimators are not considered.

Value

a list of data.frames containing the results for the respective scenarios.

See Also

[evaluate_estimator]

Examples

```
res <-evaluate_scenarios_parallel(
 score_lists = list(c(MSE(), OverestimationProbability())),
 estimator_lists =  list(c(SampleMean(), FirstStageSampleMean())),
 data_distribution_lists = list(c(Normal(FALSE), Normal(TRUE))),
 design_lists =  list(c(get_example_design())),
 mu_lists = list(c(-1, 0, 1)),
 sigma_lists = list(1)
)
```

`get_example_design` *Generate an exemplary adaptive design*

Description

The design was optimized to minimize the expected sample size under the alternative hypothesis for a one-armed trial. The boundaries are chosen to control the type I error at 0.025 for a normally distributed test statistic (i.e. known variance). For an alternative hypothesis of mu=0.4, the overall power is 80%.

Usage

```
get_example_design(two_armed = FALSE, label = NULL)
```

Arguments

`two_armed`	(logical) determins whether the design is for one- or two-armed trials.
`label`	(optional) label to be assigned to the design.

Value

an exemplary design of class `TwoStageDesign`. This object contains information about the sample size recalculation rule `n2`, the futility and efficacy boundaries `c1f` and `c1e` and the second-stage rejection boundary `c2`.

Examples

```
get_example_design()
```

`get_example_statistics` *Generate a list of estimators and p-values to use in examples*

Description

This function generates a list of objects of class `PointEstimator`, `IntervalEstimator`s, and `PValue`s to use in examples of the `analyze` function.

Usage

```
get_example_statistics(
  point_estimators = TRUE,
  interval_estimators = TRUE,
  p_values = TRUE
)
```

Arguments

`point_estimators`	logical indicating whether point estimators should be included in output list
`interval_estimators`	logical indicating whether interval estimators should be included in output list
`p_values`	logical indicating whether p-values should be included in output list

Details

Point estimators:

The following `PointEstimator`s are included:

- `SampleMean`
- `PseudoRaoBlackwell`
- `MedianUnbiasedLikelihoodRatioOrdering`
- `BiasReduced`

Confidence intervals:

The following `IntervalEstimator`s are included:

- `StagewiseCombinationFunctionOrderingCI`
- `LikelihoodRatioOrderingCI`

P-Values:

The following `PValue`s are included:

- `StagewiseCombinationFunctionOrderingPValue`
- `LikelihoodRatioOrderingPValue`

Value

a list of `PointEstimator`s, `IntervalEstimator`s and `PValue`.

Examples

```
set.seed(123)
dat <- data.frame(
  endpoint = c(rnorm(28, 0.3)),
  stage = rep(1, 28)
)
analyze(data = dat,
        statistics = list(),
        data_distribution = Normal(FALSE),
        design = get_example_design(),
        sigma = 1)

# The results suggest recruiting 32 patients for the second stage
dat <- rbind(
  dat,
  data.frame(
    endpoint = rnorm(32, mean = 0.3),
    stage = rep(2, 32)))
analyze(data = dat,
        statistics = get_example_statistics(),
        data_distribution = Normal(FALSE),
        design = get_example_design(),
        sigma = 1)
```

`get_stagewise_estimators` *Conditional representations of an estimator or p-value*

Description

This generic determines the functional representations of point and interval estimators and p-values. The functions are returned in two parts, one part to calculate the values conditional on early futility or efficacy stops (i.e. where no second stage mean and sample size is available), and one conditional on continuation to the second stage.

Usage

```
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'VirtualPointEstimator,ANY'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'VirtualPValue,ANY'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
```

```
)

## S4 method for signature 'VirtualIntervalEstimator,ANY'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'PointEstimator,Student'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'PValue,Student'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'IntervalEstimator,Student'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'VirtualPointEstimator,Student'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'VirtualIntervalEstimator,Student'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'VirtualPValue,Student'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'PointEstimator,DataDistribution'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'PValue,DataDistribution'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'IntervalEstimator,DataDistribution'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'AdaptivelyWeightedSampleMean,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MinimizePeakVariance,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'BiasReduced,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'RaoBlackwell,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'PseudoRaoBlackwell,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'RepeatedCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'LinearShiftRepeatedPValue,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MLEOrderingPValue,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'LikelihoodRatioOrderingPValue,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)
```

```
## S4 method for signature 'ScoreTestOrderingPValue,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'StagewiseCombinationFunctionOrderingPValue,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'NeymanPearsonOrderingPValue,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'NaivePValue,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'StagewiseCombinationFunctionOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
```

```
)

## S4 method for signature 'MLEOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'LikelihoodRatioOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'ScoreTestOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'NeymanPearsonOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'NaiveCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature
## 'MidpointStagewiseCombinationFunctionOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MidpointMLEOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MidpointLikelihoodRatioOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MidpointScoreTestOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MidpointNeymanPearsonOrderingCI,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature
## 'MedianUnbiasedStagewiseCombinationFunctionOrdering,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MedianUnbiasedMLEOrdering,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MedianUnbiasedLikelihoodRatioOrdering,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)

## S4 method for signature 'MedianUnbiasedScoreTestOrdering,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)
```

```
## S4 method for signature 'MedianUnbiasedNeymanPearsonOrdering,Normal'
get_stagewise_estimators(
  estimator,
  data_distribution,
  use_full_twoarm_sampling_distribution = FALSE,
  design,
  sigma,
  exact = FALSE
)
```

Arguments

`estimator`	object of class `PointEstimator`, `IntervalEstimator` or `PValue`.
`data_distribution`	object of class `Normal` or `Student`.
`use_full_twoarm_sampling_distribution`	logical indicating whether this estimator is intended to be used with the full sampling distribution in a two-armed trial.
`design`	object of class `TwoStageDesign`.
`sigma`	assumed standard deviation.
`exact`	logical indicating usage of exact n2 function.

Value

a list with the conditional functional representations (one for each stage where the trial might end) of the estimator or p-value.

Examples

```
get_stagewise_estimators(
  estimator = SampleMean(),
  data_distribution = Normal(FALSE),
  use_full_twoarm_sampling_distribution = FALSE,
  design = get_example_design(),
  sigma = 1,
  exact = FALSE
)
```

`get_statistics_from_paper` *Generate the list of estimators and p-values that were used in the paper*

Description

Generate the list of estimators and p-values that were used in the paper

Usage

```
get_statistics_from_paper(
  point_estimators = TRUE,
  interval_estimators = TRUE,
  p_values = TRUE
)
```

Arguments

`point_estimators`	logical indicating whether point estimators should be included in output list
`interval_estimators`	logical indicating whether interval estimators should be included in output list
`p_values`	logical indicating whether p-values should be included in output list

Value

a list of `PointEstimator`s, `IntervalEstimator`s and `PValue`.

Examples

```
set.seed(123)
dat <- data.frame(
  endpoint = c(rnorm(28, 0.3)),
  stage = rep(1, 28)
)
analyze(data = dat,
        statistics = list(),
        data_distribution = Normal(FALSE),
        design = get_example_design(),
        sigma = 1)

# The results suggest recruiting 32 patients for the second stage
dat <- rbind(
  dat,
  data.frame(
    endpoint = rnorm(32, mean = 0.3),
    stage = rep(2, 32)))
analyze(data = dat,
        statistics = get_example_statistics(),
        data_distribution = Normal(FALSE),
        design = get_example_design(),
        sigma = 1)
```

IntervalEstimator-class — *Interval estimators*

Description

This is the parent class for all confidence intervals implemented in this package. Currently, only confidence intervals for the parameter μ of a normal distribution are implemented. Details about the methods for calculating confidence intervals can be found in (our upcoming paper).

Usage

```
IntervalEstimator(two_sided, l1, u1, l2, u2, label)

RepeatedCI(two_sided = TRUE)

StagewiseCombinationFunctionOrderingCI(two_sided = TRUE)

MLEOrderingCI(two_sided = TRUE)

LikelihoodRatioOrderingCI(two_sided = TRUE)

ScoreTestOrderingCI(two_sided = TRUE)

NeymanPearsonOrderingCI(two_sided = TRUE, mu0 = 0, mu1 = 0.4)

NaiveCI(two_sided = TRUE)
```

Arguments

`two_sided`	logical indicating whether the confidence interval is two-sided.
`l1`	functional representation of the lower boundary of the interval in the early futility and efficacy regions.
`u1`	functional representation of the upper boundary of the interval in the early futility and efficacy regions.
`l2`	functional representation of the lower boundary of the interval in the continuation region.
`u2`	functional representation of the upper boundary of the interval in the continuation region.
`label`	name of the estimator. Used in printing methods.
`mu0`	expected value of the normal distribution under the null hypothesis.
`mu1`	expected value of the normal distribution under the null hypothesis.

Details

The implemented confidence intervals are:

- `MLEOrderingCI()`
- `LikelihoodRatioOrderingCI()`
- `ScoreTestOrderingCI()`
- `StagewiseCombinationFunctionOrderingCI()`

These confidence intervals are constructed by specifying an ordering of the sample space and finding the value of μ, such that the observed sample is the $\alpha/2$ (or $(1-\alpha/2)$) quantile of the sample space according to the chosen ordering. Some of the implemented orderings are based on the work presented in (Emerson and Fleming 1990), (Sections 8.4 in Jennison and Turnbull 1999), and (Sections 4.1.1 and 8.2.1 in Wassmer and Brannath 2016).

Value

an object of class `IntervalEstimator`. This class signals that an object can be supplied to the `evaluate_estimator` and the `analyze` functions.

References

Emerson SS, Fleming TR (1990). "Parameter estimation following group sequential hypothesis testing." *Biometrika*, **77**(4), 875–892. doi:10.2307/2337110.

Jennison C, Turnbull BW (1999). *Group Sequential Methods with Applications to Clinical Trials*, 1 edition. Chapman and Hall/CRC., New York. doi:10.1201/9780367805326.

Wassmer G, Brannath W (2016). *Group Sequential and Confirmatory Adaptive Designs in Clinical Trials*, 1 edition. Springer, Cham, Switzerland. doi:10.1007/9783319325620.

See Also

`evaluate_estimator`

Examples

```
# This is the definition of the 'naive' confidence interval for one-armed trials
IntervalEstimator(
  two_sided = TRUE,
  l1 = \(smean1, n1, sigma, ...) smean1 - qnorm(.95, sd = sigma/sqrt(n1)),
  u1 = \(smean1, n1, sigma, ...) smean1 + qnorm(.95, sd = sigma/sqrt(n1)),
 l2 = \(smean1, smean2, n1, n2, sigma, ...) smean2 - qnorm(.95, sd = sigma/sqrt(n1 + n2)),
 u2 = \(smean1, smean2, n1, n2, sigma, ...) smean2 + qnorm(.95, sd = sigma/sqrt(n1 + n2)),
  label="My custom CI")
```

`n2_extrapol` *Calculate the second-stage sample size for a design with cached spline parameters*

Description

Also extrapolates results for values outside of [c1f, c1e].

Usage

```
n2_extrapol(design, x1)
```

Arguments

`design`	an object of class `TwoStageDesignWithCache`.
`x1`	first-stage test statistic

`NormalPrior` *Normal prior distribution for the parameter mu*

Description

Normal prior distribution for the parameter mu

Usage

```
NormalPrior(mu = 0, sigma = 1)
```

Arguments

`mu`	mean of prior distribution.
`sigma`	standard deviation of the prior distribution.

Value

an object of class `NormalPrior`. This object can be supplied as the argument `mu` of the `evaluate_estimator` function to calculate performance scores weighted by a prior.

Examples

```
NormalPrior(mu = 0, sigma = 1)
```

plot,EstimatorScoreResult-method

Plot performance scores for point and interval estimators

Description

This function extract the values of mu and the score values and a facet plot with one facet per score. If the input argument is a list, the different estimators will be displayed in the same facets, differentiated by color.

Usage

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

Arguments

`x`	an output object from evaluate_estimator (`EstimatorScoreResult`) or a list of such objects (`EstimatorScoreResultList`).
`y`	unused.
`...`	additional arguments handed down to ggplot.

Value

a `ggplot` object visualizing the score values.

Examples

```
score_result1 <- evaluate_estimator(
  MSE(),
  estimator = SampleMean(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu=seq(-.75, 1.32, 0.03),
  sigma=1)
# Plotting the result of evaluate_estimator
plot(score_result1)

score_result2 <- evaluate_estimator(
  MSE(),
  estimator = AdaptivelyWeightedSampleMean(w1 = 0.8),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu=seq(-.75, 1.32, 0.03),
  sigma=1)
# Plotting a list of different score results
plot(c(score_result1, score_result2))
```

`plot,EstimatorScoreResultList-method`

Plot performance scores for point and interval estimators

Description

This function extract the values of mu and the score values and a facet plot with one facet per score. If the input argument is a list, the different estimators will be displayed in the same facets, differentiated by color.

Usage

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

Arguments

`x`	an output object from evaluate_estimator (`EstimatorScoreResult`) or a list of such objects (`EstimatorScoreResultList`).
`y`	unused.
`...`	additional arguments handed down to ggplot.

Value

a `ggplot` object visualizing the score values.

Examples

```
score_result1 <- evaluate_estimator(
  MSE(),
  estimator = SampleMean(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu=seq(-.75, 1.32, 0.03),
  sigma=1)
# Plotting the result of evaluate_estimator
plot(score_result1)

score_result2 <- evaluate_estimator(
  MSE(),
  estimator = AdaptivelyWeightedSampleMean(w1 = 0.8),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu=seq(-.75, 1.32, 0.03),
  sigma=1)
# Plotting a list of different score results
plot(c(score_result1, score_result2))
```

`plot,list-method` *Plot performance scores for point and interval estimators*

Description

This function extract the values of mu and the score values and a facet plot with one facet per score. If the input argument is a list, the different estimators will be displayed in the same facets, differentiated by color.

Usage

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

Arguments

`x`	an output object from evaluate_estimator (`EstimatorScoreResult`) or a list of such objects (`EstimatorScoreResultList`).
`y`	unused.
`...`	additional arguments handed down to ggplot.

Value

a `ggplot` object visualizing the score values.

Examples

```
score_result1 <- evaluate_estimator(
  MSE(),
  estimator = SampleMean(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu=seq(-.75, 1.32, 0.03),
  sigma=1)
# Plotting the result of evaluate_estimator
plot(score_result1)

score_result2 <- evaluate_estimator(
  MSE(),
  estimator = AdaptivelyWeightedSampleMean(w1 = 0.8),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu=seq(-.75, 1.32, 0.03),
  sigma=1)
# Plotting a list of different score results
plot(c(score_result1, score_result2))
```

plot_p *Plot p-values and implied rejection boundaries*

Description

Creates a plot of the p-values and implied rejection boundaries on a grid of values for the first and second-stage test statistics.

Usage

```
plot_p(
  estimator,
  data_distribution,
  design,
  mu = 0,
  sigma,
  boundary_color = "lightgreen",
  subdivisions = 100,
  ...
)
```

Arguments

`estimator`	object of class `PointEstimator`, `IntervalEstimator` or `PValue`.
`data_distribution`	object of class `Normal` or `Student`.
`design`	object of class `TwoStageDesign`.
`mu`	expected value of the underlying normal distribution.
`sigma`	assumed standard deviation.
`boundary_color`	color of the implied rejection boundary.
`subdivisions`	number of subdivisions per axis for the grid of test statistic values.
`...`	additional arguments handed down to ggplot

Details

When the first-stage test statistic lies below the futility threshold (c1f) or above the early efficacy threshold (c1e) of the `TwoStageDesign`, there is no second-stage test statistics. The p-values in these regions are only based on the first-stage values. For first-stage test statistic values between c1f and c1e, the first and second-stage test statistic determine the p-value.

The rejection boundary signals the line where

Value

a `ggplot` object visualizing the p-values on a grid of possible test-statistic values.

Examples

```
plot_p(estimator = StagewiseCombinationFunctionOrderingPValue(),
  data_distribution = Normal(FALSE),
  design = get_example_design(),
  mu = 0,
  sigma = 1)
```

PointEstimator-class *Point estimators*

Description

This is the parent class for all point estimators implemented in this package. Currently, only estimators for the parameter μ of a normal distribution are implemented.

Usage

```
PointEstimator(g1, g2, label)

SampleMean()

FirstStageSampleMean()

WeightedSampleMean(w1 = 0.5)

AdaptivelyWeightedSampleMean(w1 = 1/sqrt(2))

MinimizePeakVariance()

BiasReduced(iterations = 1L)

RaoBlackwell()

PseudoRaoBlackwell()

MidpointStagewiseCombinationFunctionOrderingCI()

MidpointMLEOrderingCI()

MidpointLikelihoodRatioOrderingCI()

MidpointScoreTestOrderingCI()

MidpointNeymanPearsonOrderingCI()

MedianUnbiasedStagewiseCombinationFunctionOrdering()
```

```
MedianUnbiasedMLEOrdering()

MedianUnbiasedLikelihoodRatioOrdering()

MedianUnbiasedScoreTestOrdering()

MedianUnbiasedNeymanPearsonOrdering(mu0 = 0, mu1 = 0.4)
```

Arguments

`g1`	functional representation of the estimator in the early futility and efficacy regions.
`g2`	functional representation of the estimator in the continuation region.
`label`	name of the estimator. Used in printing methods.
`w1`	weight of the first-stage data.
`iterations`	number of bias reduction iterations. Defaults to 1.
`mu0`	expected value of the normal distribution under the null hypothesis.
`mu1`	expected value of the normal distribution under the null hypothesis.

Details

Details about the point estimators can be found in (our upcoming paper).

Sample Mean (`SampleMean()`):
The sample mean is the maximum likelihood estimator for the mean and probably the 'most straightforward' of the implemented estimators.

Fixed weighted sample means (`WeightedSampleMean()`):
The first- and second-stage (if available) sample means are combined via fixed, predefined weights. See (Brannath et al. 2006) and (Section 8.3.2 in Wassmer and Brannath 2016).

Adaptively weighted sample means (`AdaptivelyWeightedSampleMean()`):
The first- and second-stage (if available) sample means are combined via a combination of fixed and adaptively modified weights that depend on the standard error. See (Section 8.3.4 in Wassmer and Brannath 2016).

Minimizing peak variance in adaptively weighted sample means (`MinimizePeakVariance()`):

For this estimator, the weights of the adaptively weighted sample mean are chosen to minimize the variance of the estimator for the value of μ which maximizes the expected sample size.

(Pseudo) Rao-Blackwell estimators (`RaoBlackwell` and `PseudoRaoBlackwell`):
The conditional expectation of the first-stage sample mean given the overall sample mean and the second-stage sample size. See (Emerson and Kittelson 1997).

A bias-reduced estimator (`BiasReduced()`):
This estimator is calculated by subtracting an estimate of the bias from the MLE. See (Whitehead 1986).

Median-unbiased estimators:

The implemented median-unbiased estimators are:

- `MedianUnbiasedMLEOrdering()`
- `MedianUnbiasedLikelihoodRatioOrdering()`
- `MedianUnbiasedScoreTestOrdering()`
- `MedianUnbiasedStagewiseCombinationFunctionOrdering()`

These estimators are constructed by specifying an ordering of the sample space and finding the value of μ, such that the observed sample is the median of the sample space according to the chosen ordering. Some of the implemented orderings are based on the work presented in (Emerson and Fleming 1990), (Sections 8.4 in Jennison and Turnbull 1999), and (Sections 4.1.1 and 8.2.1 in Wassmer and Brannath 2016).

Value

an object of class `PointEstimator`. This class signals that an object can be supplied to the `evaluate_estimator` and the `analyze` functions.

References

Brannath W, König F, Bauer P (2006). "Estimation in flexible two stage designs." *Statistics in Medicine*, **25**(19), 3366-3381. doi:10.1002/sim.2258.

Emerson SS, Fleming TR (1990). "Parameter estimation following group sequential hypothesis testing." *Biometrika*, **77**(4), 875–892. doi:10.2307/2337110.

Emerson SS, Kittelson JM (1997). "A computationally simpler algorithm for the UMVUE of a normal mean following a group sequential trial." *Biometrics*, **53**(1), 365–369. doi:10.2307/2533122.

Jennison C, Turnbull BW (1999). *Group Sequential Methods with Applications to Clinical Trials*, 1 edition. Chapman and Hall/CRC., New York. doi:10.1201/9780367805326.

Wassmer G, Brannath W (2016). *Group Sequential and Confirmatory Adaptive Designs in Clinical Trials*, 1 edition. Springer, Cham, Switzerland. doi:10.1007/9783319325620.

Whitehead J (1986). "On the bias of maximum likelihood estimation following a sequential test." *Biometrika*, **73**(3), 573–581. doi:10.2307/2336521.

See Also

`evaluate_estimator`

Examples

```
PointEstimator(g1 = \(smean1, ...) smean1,g2 = \(smean2, ...) smean2, label="My custom estimator")
```

PValue-class *P-values*

Description

This is the parent class for all p-values implemented in this package. Details about the methods for calculating p-values can be found in (our upcoming paper).

Usage

```
PValue(g1, g2, label)

LinearShiftRepeatedPValue(wc1f = 0, wc1e = 1/2, wc2 = 1/2)

MLEOrderingPValue()

LikelihoodRatioOrderingPValue()

ScoreTestOrderingPValue()

StagewiseCombinationFunctionOrderingPValue()

NeymanPearsonOrderingPValue(mu0 = 0, mu1 = 0.4)

NaivePValue()
```

Arguments

`g1`	functional representation of the p-value in the early futility and efficacy regions.
`g2`	functional representation of the p-value in the continuation region.
`label`	name of the p-value. Used in printing methods.
`wc1f`	slope of futility boundary change.
`wc1e`	slope of efficacy boundary change.
`wc2`	slope of c2 boundary change.
`mu0`	expected value of the normal distribution under the null hypothesis.
`mu1`	expected value of the normal distribution under the null hypothesis.

Details

The implemented p-values are:

- `MLEOrderingPValue()`
- `LikelihoodRatioOrderingPValue()`
- `ScoreTestOrderingPValue()`
- `StagewiseCombinationFunctionOrderingPValue()`

The p-values are calculated by specifying an ordering of the sample space calculating the probability that a random sample under the null hypothesis is larger than the observed sample. Some of the implemented orderings are based on the work presented in (Emerson and Fleming 1990), (Sections 8.4 in Jennison and Turnbull 1999), and (Sections 4.1.1 and 8.2.1 in Wassmer and Brannath 2016).

Value

an object of class `PValue`. This class signals that an object can be supplied to the `analyze` function.

References

Emerson SS, Fleming TR (1990). "Parameter estimation following group sequential hypothesis testing." *Biometrika*, **77**(4), 875–892. doi:10.2307/2337110.

Jennison C, Turnbull BW (1999). *Group Sequential Methods with Applications to Clinical Trials*, 1 edition. Chapman and Hall/CRC., New York. doi:10.1201/9780367805326.

Wassmer G, Brannath W (2016). *Group Sequential and Confirmatory Adaptive Designs in Clinical Trials*, 1 edition. Springer, Cham, Switzerland. doi:10.1007/9783319325620.

See Also

plot_p

Examples

```
# This is the definition of a 'naive' p-value based on a Z-test for a one-armed trial
PValue(
  g1 = \(smean1, n1, sigma, ...) pnorm(smean1*sqrt(n1)/sigma, lower.tail=FALSE),
  g2 = \(smean1, smean2, n1, n2, ...) pnorm((n1 * smean1 + n2 * smean2)/(n1 + n2) *
                                       sqrt(n1+n2)/sigma, lower.tail=FALSE),
  label="My custom p-value")
```

`Statistic-class` *Statistics and Estimators of the adestr package*

Description

The `Statistic` class is a parent class for the classes `Estimator` and `PValue`. The `Estimator` class is a parent for the classes `PointEstimator` and `ConfidenceInterval`.

Arguments

`label`	name of the statistic. Used in printing methods.

Details

The function `analyze` can be used to calculate the value of a `Statistic` for a given dataset.

The function `evaluate_estimator` can be used to evaluate distributional quantities of an `Estimator` like the `MSE` for a `PointEstimator` or the `Coverage` for a `ConfidenceInterval`.

Value

An object of class `Statistic`. This class signals that an object can be supplied to the `analyze` function.

See Also

`PointEstimator` `ConfidenceInterval` `PValue`

`analyze` `evaluate_estimator`

`EstimatorScore`

`TwoStageDesignWithCache` *TwoStageDesignWithCache constructor function*

Description

Creates an object of class `TwoStageDesignWithCache`. This object stores the precalculated spline paramters of the `n2` and `c2` functions, which allows for quicker evaluation.

Usage

```
TwoStageDesignWithCache(design)
```

Arguments

`design`	an object of class TwoStageDesign

`UniformPrior` *Uniform prior distribution for the parameter mu*

Description

Uniform prior distribution for the parameter mu

Usage

```
UniformPrior(min = -1, max = 1)
```

Arguments

`min`	minimum of support interval.
`max`	maximum of support interval.

Value

an object of class `UniformPrior`. This object can be supplied as the argument `mu` of the `evaluate_estimator` function to calculate performance scores weighted by a prior.

Examples

```
UniformPrior(min = -1, max = 1)
```

Index