Package: baskexact (via r-universe)

October 29, 2026

Type Package

Title Analytical Calculation of Basket Trial Operating Characteristics

Version 1.0.1

Description Analytically calculates the operating characteristics of single-stage and two-stage basket trials with equal sample sizes using the power prior design by Baumann et al. (2024) <doi:10.48550/arXiv.2309.06988> and the design by Fujikawa et al. (2020) <doi:10.1002/bimj.201800404>.

License GPL (>= 3)

URL https://github.com/lbau7/baskexact

Imports arrangements, doFuture, extraDistr, foreach, ggplot2, methods, Rcpp

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

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

Collate 'RcppExports.R' 'class.R' 'adjust_lambda.R' 'analysis.R' 'basket_test.R' 'baskexact-package.R' 'borrowing.R' 'check.R' 'ecd.R' 'ess.R' 'estim.R' 'globalweights.R' 'helper.R' 'interim.R' 'monotonicity.R' 'opt_design.R' 'plot.R' 'pow.R' 'rejection_probabilities.R' 'toer.R' 'validate.R' 'weights.R'

NeedsCompilation yes

Author Lukas Baumann [aut, cre] (<https://orcid.org/0000-0001-7931-7470>)

Maintainer Lukas Baumann <baumann@imbi.uni-heidelberg.de>

Repository https://cran.r-universe.dev

Date/Publication 2024-04-10 02:34:46 UTC

RemoteUrl https://github.com/cran/baskexact

RemoteRef HEAD

RemoteSha 6ce99eafc9fa698ff22bbdb049d824e4b9c310f9

Contents

adjust_lambda . . . 2
basket_test . . . 4
check_mon_between . . . 6
check_mon_within . . . 7
ecd . . . 9
ess . . . 11
estim . . . 12
get_scenarios . . . 14
globalweights_diff . . . 15
globalweights_fix . . . 16
interim_posterior . . . 16
interim_postpred . . . 18
OneStageBasket-class . . . 19
opt_design . . . 20
plot_weights . . . 22
pow . . . 23
setupOneStageBasket . . . 25
setupTwoStageBasket . . . 26
toer . . . 26
TwoStageBasket-class . . . 28
weights_cpp . . . 29
weights_fujikawa . . . 31
weights_jsd . . . 33
weights_mml . . . 35
weights_pool . . . 36
weights_separate . . . 37

Index **39**

`adjust_lambda` *Adjust Lambda*

Description

Finds the value for `lambda` such that the family wise error rate is protected at level `alpha`.

Usage

```
adjust_lambda(design, ...)

## S4 method for signature 'OneStageBasket'
adjust_lambda(
  design,
  alpha = 0.025,
  p1 = NULL,
  n,
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  prec_digits,
  ...
)

## S4 method for signature 'TwoStageBasket'
adjust_lambda(
  design,
  alpha = 0.025,
  p1 = NULL,
  n,
  n1,
  interim_fun,
  interim_params = list(),
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  prec_digits,
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
alpha	The one-sided signifance level.
p1	Probabilities under the alternative hypothesis. If length(p1) == 1, then this is a common probability for all baskets. If is.null(p1) then the type 1 error rate under the global null hypothesis is computed.
n	The sample size per basket.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.

globalweight_params	A list of tuning parameters specific to globalweight_fun.
prec_digits	Number of decimal places that are considered when adjusting lambda.
n1	The sample size per basket for the interim analysis in case of a two-stage design.
interim_fun	Which type of interim analysis should be conducted in case of a two-stage design.
interim_params	A list of tuning parameters specific to interim_fun.

Details

adjust_alpha finds the greatest value with prec_digits for lambda which controls the family wise error rate at level alpha (one-sided). A combination of the uniroot function followed by a grid search is used to finde the correct value for lambda.

Value

The greatest value with prec_digits decimal places for lambda which controls the family wise error rate at level alpha (one-sided) and the exact family wise error rate for this value of lambda.

Methods (by class)

- adjust_lambda(OneStageBasket): Adjust lambda for a single-stage design.
- adjust_lambda(TwoStageBasket): Adjust lambda for a two-stage design.

Examples

```
design <- setupOneStageBasket(k = 3, shape1 = 1, shape2 = 1, p0 = 0.2)
adjust_lambda(design = design, alpha = 0.025, n = 15,
  weight_fun = weights_fujikawa, prec_digits = 4)
```

---

basket_test *Test for the Results of a Basket Trial*

Description

basket_test evaluates the results of a basket trial and calculates the posterior distributions with and without borrowing.

Usage

```
basket_test(design, ...)

## S4 method for signature 'OneStageBasket'
basket_test(
  design,
  n,
  r,
```

```
  lambda,
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  details = TRUE,
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
r	The vector of observed responses.
lambda	The posterior probability threshold. See details for more information.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
details	Whether a detailed list of results or only the vector of posterior probabilities is returned.

Value

If details = TRUE: A list, including matrices of the weights that are used for borrowing, posterior distribution parameters for all baskets without and with borrowing, as well as the posterior probabilities for all baskets without and with borrowing. If details = FALSE: The posterior probabilities for all baskets with borrowing.

Methods (by class)

- basket_test(OneStageBasket): Testing for a single-stage basket design.

Examples

```
design <- setupOneStageBasket(k = 3, shape1 = 1, shape2 = 1, p0 = 0.2)
basket_test(design = design, n = 24, r = c(5, 9, 10), lambda = 0.99,
  weight_fun = weights_fujikawa)
```

check_mon_between *Check Between-Trial Monotonicity*

Description

Checks whether the between-trial monotonicity condition holds.

Usage

```
check_mon_between(design, ...)

## S4 method for signature 'OneStageBasket'
check_mon_between(
  design,
  n,
  lambda,
  weight_fun,
  weight_params = list(),
  details = TRUE,
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
lambda	The posterior probability threshold. See details for more information.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
details	Whether the cases where the monotonicity condition is violated should be returned, in case there are any.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.

Details

check_mon_between checks whether the between-trial monotonicity condition holds. For a single-stage design with equal prior distributions and equal sample sizes for each basket this condition states that there are no cases where at least one null hypothesis is rejected when when there is a case with an equal or higher number of responses in each basket for which no null hypothesis is rejected.

If prune = TRUE then the baskets with an observed number of baskets smaller than the pooled critical value are not borrowed from. The pooled critical value is the smallest integer c for which all null hypotheses can be rejected if the number of responses is exactly c for all baskets.

The function is vectorized, such that vectors can be specified in weight_params and globalweight_params.

Value

If details = FALSE then only a logical value is returned. If details = TRUE then if there are any cases where the between-trial monotonicity condition is violated, a list of theses cases and their results are returned.

Methods (by class)

- check_mon_between(OneStageBasket): Between-trial monotonicity condition for a single-stage design.

References

Baumann, L., Krisam, J., & Kieser, M. (2022). Monotonicity conditions for avoiding counterintuitive decisions in basket trials. Biometrical Journal, 64(5), 934-947.

Examples

```
design <- setupOneStageBasket(k = 4, shape1 = 1, shape2 = 1, p0 = 0.2)

# Without vectorization, with details
check_mon_between(design = design, n = 24, lambda = 0.99,
  weight_fun = weights_fujikawa, weight_params = list(epsilon = 3,
    tau = 0), details = TRUE)

# Vectorized
check_mon_between(design = design, n = 24, lambda = 0.99,
  weight_fun = weights_fujikawa,
  weight_params = list(epsilon = c(0.5, 1),  tau = c(0, 0.2, 0.3)),
  globalweight_fun = globalweights_fix,
  globalweight_params = list(w = c(0.5, 0.7)))
```

---

check_mon_within *Check Within-Trial Monotonicity*

Description

Checks whether the within-trial monotonicity condition holds.

Usage

```
check_mon_within(design, ...)

## S4 method for signature 'OneStageBasket'
check_mon_within(
  design,
  n,
  lambda,
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  details = TRUE,
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
lambda	The posterior probability threshold. See details for more information.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
details	Whether the cases where the monotonicity condition is violated should be returned, in case there are any.

Details

check_mon_within checks whether the within-trial monotonicity condition holds. For a single-stage design with equal prior distributions and equal sample sizes for each basket this condition states that there are no cases where the null hypothesis of a basket is rejected when there is at least one other basket with more observed responses for which the null hypothesis cannot be rejected.

If prune = TRUE then the baskets with an observed number of baskets smaller than the pooled critical value are not borrowed from. The pooled critical value is the smallest integer c for which all null hypotheses can be rejected if the number of responses is exactly c for all baskets.

The function is vectorized, such that vectors can be specified in weight_params and globalweight_params.

Value

If details = FALSE then only a logical value is returned. If details = TRUE then if there are any cases where the within-trial monotonicity condition is violated, a list of these cases and their results are returned. If at least one tuning parameter is a vector, then an array that shows for which

combination of parameters the within-trial monotonicity condition holds. In this case, the argument `details` is ignored.

Methods (by class)

- `check_mon_within(OneStageBasket)`: Within-trial monotonicity condition for a single-stage design.

References

Baumann, L., Krisam, J., & Kieser, M. (2022). Monotonicity conditions for avoiding counterintuitive decisions in basket trials. Biometrical Journal, 64(5), 934-947.

Examples

```
design <- setupOneStageBasket(k = 4, shape1 = 1, shape2 = 1, p0 = 0.2)

# Without vectorization, with details
design <- setupOneStageBasket(k = 4, shape1 = 1, shape2 = 1, p0 = 0.2)
check_mon_within(design = design, n = 24, lambda = 0.99,
  weight_fun = weights_fujikawa, weight_params = list(epsilon = 0.5,
   tau = 0), details = TRUE)

# Vectorized
check_mon_within(design = design, n = 24, lambda = 0.99,
  weight_fun = weights_fujikawa,
  weight_params = list(epsilon = c(0.5, 1),  tau = c(0, 0.2, 0.3)),
  globalweight_fun = globalweights_fix,
  globalweight_params = list(w = c(0.5, 0.7)))
```

---

ecd *Expected number of correct decisions*

Description

Computes the expected number of correct decisions of a basket trial.

Usage

```
ecd(design, ...)

## S4 method for signature 'OneStageBasket'
ecd(
  design,
  p1 = NULL,
  n,
  lambda,
  weight_fun,
  weight_params = list(),
```

```
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)

## S4 method for signature 'TwoStageBasket'
ecd(
  design,
  p1 = NULL,
  n,
  n1,
  lambda,
  interim_fun,
  interim_params = list(),
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
p1	Probabilities under the alternative hypothesis. If length(p1) == 1, then this is a common probability for all baskets.
n	The sample size per basket.
lambda	The posterior probability threshold. See details for more information.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
n1	The sample size per basket for the interim analysis in case of a two-stage design.
interim_fun	Which type of interim analysis should be conducted in case of a two-stage design.
interim_params	A list of tuning parameters specific to interim_fun.

Details

Computes the expected number of correction decisions, i.e. the expected number of actually active baskets that are declared active and actually inactive baskets that are declared inactive.

Value

A numeric value.

Methods (by class)

- `ecd(OneStageBasket)`: Expected number of correction decisions for a single-stage basket design.
- `ecd(TwoStageBasket)`: Expected number of correction decisions for a two-stage basket design.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
ecd(design = design, p1 = c(0.5, 0.2, 0.2), n = 20, lambda = 0.99,
weight_fun = weights_fujikawa)
```

---

ess *Expected Sample Size*

Description

Computes the expected sample size of a two-stage basket trial.

Usage

```
ess(design, ...)

## S4 method for signature 'TwoStageBasket'
ess(
  design,
  p1 = NULL,
  n,
  n1,
  lambda,
  interim_fun,
  interim_params = list(),
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
p1	Probabilities under the alternative hypothesis. If length(p1) == 1, then this is a common probability for all baskets. If is.null(p1) then the type 1 error rate under the global null hypothesis is computed.
n	The sample size per basket.
n1	The sample size per basket for the interim analysis in case of a two-stage design.
lambda	The posterior probability threshold. See details for more information.
interim_fun	Which type of interim analysis should be conducted in case of a two-stage design.
interim_params	A list of tuning parameters specific to interim_fun.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.

Methods (by class)

- ess(TwoStageBasket): Expected sample size for two-stage basket design.

Examples

```
design <- setupTwoStageBasket(k = 3, p0 = 0.2)
ess(design, n = 20, n1 = 10, lambda = 0.99, weight_fun = weights_fujikawa,
  interim_fun = interim_postpred)
```

---

estim *Posterior Mean and Mean Squared Error*

Description

Computes the posterior mean and the mean squared error of a basket trial design.

Usage

```
estim(design, ...)

## S4 method for signature 'OneStageBasket'
estim(
  design,
  p1,
```

```
  n,
  lambda = NULL,
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)

## S4 method for signature 'TwoStageBasket'
estim(
  design,
  p1,
  n,
  n1,
  lambda = NULL,
  interim_fun,
  interim_params = list(),
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
p1	Probabilities under the alternative hypothesis. If length(p1) == 1, then this is a common probability for all baskets.
n	The sample size per basket.
lambda	The posterior probability threshold. See details for more information.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
n1	The sample size per basket for the interim analysis in case of a two-stage design.
interim_fun	Which type of interim analysis should be conducted in case of a two-stage design.
interim_params	A list of tuning parameters specific to interim_fun.

Value

A list containing means of the posterior distributions and the mean squared errors for all baskets.

Methods (by class)

- `estim(OneStageBasket)`: Posterior mean and mean squared error for a single-stage basket design.
- `estim(TwoStageBasket)`: Posterior mean and mean squared error for a two-stage basket design.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
estim(design = design, p1 = c(0.2, 0.2, 0.5), n = 15,
  weight_fun = weights_fujikawa)
```

---

get_scenarios *Create a Scenario Matrix*

Description

Creates a default scenario matrix.

Usage

```
get_scenarios(design, p1)
```

Arguments

`design`	An object of class `Basket` created by `setupOneStageBasket` or `setupTwoStageBasket`.
`p1`	Probabilitiy under the alternative hypothesis.

Details

`get_scenarios` creates a default scenario matrix that can be used for `opt_design`. The function creates `k + 1` scenarios, from a global null to a global alternative scenario.

Value

A matrix with `k` rows and `k + 1` columns.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
get_scenarios(design = design, p1 = 0.5)
```

`globalweights_diff` *Global Weights Based on Response Rate Differences*

Description

Global Weights Based on Response Rate Differences

Usage

```
globalweights_diff(n, r, eps_global = 1, w = 1)
```

Arguments

`n`	The sample size per basket.
`r`	Vector of responses.
`eps_global`	A tuning parameter that determines the amount of borrowing. A higher value leads to a smaller weight and therefore less borrowing when the heterogeneity between the results in the baskets increases.
`w`	A fixed probability between 0 and 1. `w` is multiplied to the weight.

Details

`globalweights_diff` calculates a weight based on the heterogeneity of the response rates of all baskets that is multiplied to the pairwise weights calculated with the function that is passed to `weight_fun`. The weight is 1 when the number of responses is identical in all baskets and 0 if the response rates are an equidistant sequence from 0 to 1. If the maximum weight should be smaller than 1, `w` can be set to a smaller value.

Value

A numeric value.

Examples

```
globalweights_diff(n = 20, r = c(1, 3, 5), eps_global = 2)
```

globalweights_fix *Fixed Global Weights*

Description

Fixed Global Weights

Usage

```
globalweights_fix(n, r, w)
```

Arguments

n	The sample size per basket.
r	Vector of responses.
w	Fixed number with wich all weights are multiplied

Value

A numeric value.

Examples

```
globalweights_fix(n = 20, r = c(1, 3, 5), w = 0.5)
```

interim_posterior *Interim analysis based on the posterior probability*

Description

Conducts an interim analysis based on the posterior probability.

Usage

```
interim_posterior(design, ...)

## S4 method for signature 'TwoStageBasket'
interim_posterior(
  design,
  n1,
  r1,
  weight_mat,
  globalweight_fun = NULL,
  globalweight_params = list(),
  prob_futstop = 0.1,
  prob_effstop = 0.9,
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n1	The sample size per basket for the interim analysis in case of a two-stage design.
r1	Vector of responses after the interim analysis.
weight_mat	The matrix with all weights. Automatically calculated in the functions to which interim_postpred is passed.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
prob_futstop	Probability cut-off for stopping for futility.
prob_effstop	Probability cut-off for stopping for efficacy.

Details

interim_posterior conducts an interim analysis with possible stop for efficacy and futility based on the posterior probability. If the posterior probability is less than prob_fustop the basket is stopped for futility, if the posterior probability is greater than prob_effstop the basket is stopped for efficacy. If prob_fustop = 0 or prob_effstop = 1 then no futility-stop and no efficacy stop is possible, respectively.

The function is generally not called by the user but passed to another function such as toer and pow to specify which interim analysis is conducted.

Value

A vector with a length equal to the number of baskets with elements -1, 0 or 1 where -1 means stop for futility, 0 means continuation and 1 means stop for efficacy.

Methods (by class)

- interim_posterior(TwoStageBasket): Interim analysis based on the posterior probabilty for two-stage basket designs.

Examples

```
design <- setupTwoStageBasket(k = 3, p0 = 0.2)
toer(design, n = 20, n1 = 10, lambda = 0.99, weight_fun = weights_fujikawa,
  interim_fun = interim_posterior, interim_params = list(prob_futstop = 0.05,
    prob_effstop = 0.95))
```

`interim_postpred` *Interim analysis based on the posterior predictive probability*

Description

Conducts an interim analysis based on the posterior predictive probability.

Usage

```
interim_postpred(design, ...)

## S4 method for signature 'TwoStageBasket'
interim_postpred(
  design,
  n,
  n1,
  r1,
  lambda,
  weight_mat,
  globalweight_fun = NULL,
  globalweight_params,
  prob_futstop = 0.1,
  prob_effstop = 0.9,
  ...
)
```

Arguments

`design`	An object of class `Basket` created by `setupOneStageBasket` or `setupTwoStageBasket`.
`...`	Further arguments.
`n`	The sample size per basket.
`n1`	The sample size per basket for the interim analysis in case of a two-stage design.
`r1`	Vector of responses after the interim analysis.
`lambda`	The posterior probability threshold. See details for more information.
`weight_mat`	The matrix with all weights. Automatically calculated in the functions to which `interim_postpred` is passed.
`globalweight_fun`	Which function should be used to calculate the global weights.
`globalweight_params`	A list of tuning parameters specific to `globalweight_fun`.
`prob_futstop`	Probability cut-off for stopping for futility.
`prob_effstop`	Probability cut-off for stopping for efficacy.

Details

interim_postpred conducts an interim analysis with possible stop for efficacy and futility based on the posterior predictive probability. If the posterior predictive probability is less than prob_fustop the basket is stopped for futility, if the posterior predictive probability is greater than prob_effstop the basket is stopped for efficacy. If prob_fustop = 0 or prob_effstop = 1 then no futility-stop and no efficacy stop is possible, respectively.

The function is generally not called by the user but passed to another function such as toer and pow to specify which interim analysis is conducted.

Value

A vector with a length equal to the number of baskets with elements -1, 0 or 1 where -1 means stop for futility, 0 means continuation and 1 means stop for efficacy.

Methods (by class)

- interim_postpred(TwoStageBasket): Interim analysis based on the posterior predictive probabilty for two-stage basket designs.

Examples

```
design <- setupTwoStageBasket(k = 3, p0 = 0.2)
toer(design, n = 20, n1 = 10, lambda = 0.99, interim_fun = interim_postpred,
  weight_fun = weights_fujikawa)
```

OneStageBasket-class *Class OneStageBasket*

Description

OneStageBasket is an S4 class. An object of this class contains the most important design features of a single-stage basket trial.

Details

This class implements a single-stage basket trial based on the power prior design or the design proposed by Fujikawa et al. In these designs, information is borrowed between baskets by calculating weights that reflect the similarity between the baskets (and optionally the overall heterogeneity). Posterior distributions for each basket are beta distributions where the parameters are found by adding weighted sums of the observed responses and non-responses in each basket to the prior parameters (or in case of Fujikawa's design by calculating weighted sums of the individual posterior distributions).

Currently only common prior distributions and a common null hypothesis are supported.

Slots

`k` The number of baskets.

`shape1` First common shape parameter of the beta prior.

`shape2` Second common shape parameter of the beta prior.

`p0` A common probability under the null hypothesis.

References

Baumann, L., Sauer, L., & Kieser, M. (2024). A basket trial design based on power priors. arXiv:2309.06988.

Fujikawa, K., Teramukai, S., Yokota, I., & Daimon, T. (2020). A Bayesian basket trial design that borrows information across strata based on the similarity between the posterior distributions of the response probability. Biometrical Journal, 62(2), 330-338.

`opt_design` *Optimize a Basket Design*

Description

Finds the optimal tuning parameters using grid search.

Usage

```
opt_design(design, ...)

## S4 method for signature 'OneStageBasket'
opt_design(
  design,
  n,
  alpha,
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  scenarios,
  prec_digits,
  ...
)

## S4 method for signature 'TwoStageBasket'
opt_design(
  design,
  n,
  n1,
  alpha,
```

```
  interim_fun,
  interim_params = list(),
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  scenarios,
  prec_digits,
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
alpha	The one-sided signifance level.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
scenarios	A matrix of response rate scenarios. Each column corresponds to a scenario and each row corresponds to a basket. A default scenario matrix can be created with get_scenarios.
prec_digits	Number of decimal places that are considered when adjusting lambda.
n1	The sample size per basket for the interim analysis in case of a two-stage design.
interim_fun	Which type of interim analysis should be conducted in case of a two-stage design.
interim_params	A list of tuning parameters specific to interim_fun.

Details

opt_design finds the optimal combination of tuning parameter values from a the set of tuning paramters that is passed to the function. The objective function for the optimization is the mean of the expected number of correct decisions (ECD) under the passed scenarios, with the constraint that the type 1 error under the global null hypothesis must be below alpha.

Value

A matrix with the ECDs under all scenarios and the mean ECD for all combinations of tuning parameter values. The matrix is sorted decreasingly by the mean ECD.

Methods (by class)

- opt_design(OneStageBasket): Optimize a single-stage basket design.
- opt_design(TwoStageBasket): Optimize a two-stage basket design.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
opt_design(design = design, n = 10, alpha = 0.05,
  weight_fun = weights_fujikawa, weight_params = list(epsilon = c(1, 2),
  tau = c(0, 0.5)), scenarios = get_scenarios(design, 0.5), prec_digits = 3)
```

---

plot_weights *Plot Weight Functions*

Description

Plot Weight Functions

Usage

```
plot_weights(design, ...)

## S4 method for signature 'OneStageBasket'
plot_weights(design, n, r1, weight_fun, weight_params = list(), ...)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
r1	Number of responses in one basket
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.

Details

The design object is only used for the prior parameters, which affect the weights of some weight functions.

Value

A plot.

Methods (by class)

- plot_weights(OneStageBasket): Plot weights for a single-stage basket trials

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2, shape1 = 1, shape2 = 1)
plot_weights(design = design, n = 20, r1 = 10, weight_fun = weights_jsd)
```

---

pow *Power*

Description

Computes the exact power for a basket trial.

Usage

```
pow(design, ...)

## S4 method for signature 'OneStageBasket'
pow(
  design,
  p1,
  n,
  lambda,
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  results = c("ewp", "group"),
  ...
)

## S4 method for signature 'TwoStageBasket'
pow(
  design,
  p1,
  n,
  n1,
  lambda,
  interim_fun,
  interim_params = list(),
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  results = c("ewp", "group"),
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
p1	Probabilities under the alternative hypothesis. If length(p1) == 1, then this is a common probability for all baskets.
n	The sample size per basket.
lambda	The posterior probability threshold. See details for more information.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
results	Whether only the experimentwise power (option ewp) or also the rejection probabilities per group (option group) should be returned.
n1	The sample size per basket for the interim analysis in case of a two-stage design.
interim_fun	Which type of interim analysis should be conducted in case of a two-stage design.
interim_params	A list of tuning parameters specific to interim_fun.

Details

pow computes the exact experimentwise power and the exact rejection probabilities per group. The experimentwise power is the probability to reject at least one null hypothesis for a basket with p1 > p0. The rejection probabilities correspond to the type 1 error rate for baskets with p1 = p0 and to the power for baskets with p1 > p0.

If prune = TRUE then the baskets with an observed number of baskets smaller than the pooled critical value are not borrowed from. The pooled critical value is the smallest integer c for which all null hypotheses can be rejected if the number of responses is exactly c for all baskets.

This method is implemented for the class OneStageBasket.

Value

If results = "ewp" then the experimentwise power is returned as a numeric value. If results = "group" then a list with the rejection probabilities per group and the experimentwise power is returned. For baskets with p1 = p0 the rejection probabilities corresponds to the type 1 error rate, for baskets with p1 > p0 the rejection probabilities corresponds to the power.

Methods (by class)

- pow(OneStageBasket): Power for a single-stage basket design.
- pow(TwoStageBasket): Power for a two-stage basket design.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
pow(design, p1 = c(0.2, 0.5, 0.5), n = 15, lambda = 0.99,
  weight_fun = weights_fujikawa, weight_params = list(epsilon = 2, tau = 0))
```

---

setupOneStageBasket *Setup OneStageBasket*

Description

Creates an object of class OneStageBasket.

Usage

```
setupOneStageBasket(k, shape1 = 1, shape2 = 1, p0)
```

Arguments

k	The number of baskets.
shape1	First common shape parameter of the beta prior.
shape2	Second common shape parameter of the beta prior.
p0	A common probability under the null hypothesis.

Details

A OneStageBasket object contains the most important design features of a basket trial. Currently only common prior distributions and a common null hypothesis are supported.

Value

An S4 object of class OneStageBasket.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
```

setupTwoStageBasket *Setup TwoStageBasket*

Description

Creates an object of class TwoStageBasket.

Usage

```
setupTwoStageBasket(k, shape1 = 1, shape2 = 1, p0)
```

Arguments

k	The number of baskets.
shape1	First common shape parameter of the beta prior.
shape2	Second common shape parameter of the beta prior.
p0	A common probability under the null hypothesis.

Details

A TwoStageBasket object contains the most important design features of a basket trial. Currently only common prior distributions and a common null hypothesis are supported.

Value

An S4 object of class TwoStageBasket.

Examples

```
design <- setupTwoStageBasket(k = 3, p0 = 0.2)
```

toer *Type 1 Error Rate*

Description

Computes the exact family wise type 1 error rate of a basket trial .

Usage

```
toer(design, ...)

## S4 method for signature 'OneStageBasket'
toer(
  design,
  p1 = NULL,
  n,
  lambda,
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  results = c("fwer", "group"),
  ...
)

## S4 method for signature 'TwoStageBasket'
toer(
  design,
  p1 = NULL,
  n,
  n1,
  lambda,
  interim_fun,
  interim_params = list(),
  weight_fun,
  weight_params = list(),
  globalweight_fun = NULL,
  globalweight_params = list(),
  results = c("fwer", "group"),
  ...
)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
p1	Probabilities under the alternative hypothesis. If length(p1) == 1, then this is a common probability for all baskets. If is.null(p1) then the type 1 error rate under the global null hypothesis is computed.
n	The sample size per basket.
lambda	The posterior probability threshold. See details for more information.
weight_fun	Which function should be used to calculate the pairwise weights.
weight_params	A list of tuning parameters specific to weight_fun.
globalweight_fun	Which function should be used to calculate the global weights.

`globalweight_params`
: A list of tuning parameters specific to `globalweight_fun`.

`results`
: Whether only the family wise error rate (option `fwer`) or also the rejection probabilities per group (option `group`) should be returned.

`n1`
: The sample size per basket for the interim analysis in case of a two-stage design.

`interim_fun`
: Which type of interim analysis should be conducted in case of a two-stage design.

`interim_params`
: A list of tuning parameters specific to `interim_fun`.

Details

`toer` computes the exact family wise type 1 error rate and the exact rejection probabilities per group. The family wise type 1 error rate is the probability to reject at least one null hypothesis for a basket with p1 = p0. If all p1 > p0 then the family wise type 1 error rate under the global null hypothesis is computed. The rejection probabilities correspond to the type 1 error rate for baskets with p1 = p0 and to the power for baskets with p1 > p0.

Value

If `results = "fwer"` then the family wise type 1 error rate is returned as a numeric value. If `results = "group"` then a list with the rejection probabilities per group and the family wise type 1 error rate is returned. If all p1 > p0 then the family wise type 1 error rate is calculated under the global null hypothesis. For baskets with p1 = p0 the rejection probabilities corresponds to the type 1 error rate, for baskets with p1 > p0 the rejection probabilities corresponds to the power.

Methods (by class)

- `toer(OneStageBasket)`: Type 1 error rate for a single-stage basket design.
- `toer(TwoStageBasket)`: Type 1 error rate for two-stage basket design.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
toer(design, n = 15, lambda = 0.99, weight_fun = weights_fujikawa)
```

TwoStageBasket-class *Class TwoStageBasket*

Description

TwoStageBasket is an S4 class. An object of this class contains the most important design features of a two-stage basket trial.

Details

This class implements a two-stage basket trial based on the power prior design or the design proposed by Fujikawa et al. In these designs, information is borrowed between baskets by calculating weights that reflect the similarity between the baskets (and optionally the overall heterogeneity). Posterior distributions for each basket are beta distributions where the parameters are found by adding weighted sums of the observed responses and non-responses in each basket to the prior parameters (or in case of Fujikawa's design by calculating weighted sums of the individual posterior distributions).

Slots

`k` The number of baskets.

`shape1` First common shape parameter of the beta prior.

`shape2` Second common shape parameter of the beta prior.

`p0` A common probability under the null hypothesis.

References

Baumann, L., Sauer, L., & Kieser, M. (2024). A basket trial design based on power priors. arXiv:2309.06988.

Fujikawa, K., Teramukai, S., Yokota, I., & Daimon, T. (2020). A Bayesian basket trial design that borrows information across strata based on the similarity between the posterior distributions of the response probability. Biometrical Journal, 62(2), 330-338.

`weights_cpp` *Weights Based on the Calibrated Power Prior*

Description

Weights Based on the Calibrated Power Prior

Usage

```
weights_cpp(design, ...)

## S4 method for signature 'OneStageBasket'
weights_cpp(
  design,
  n,
  a = 1,
  b = 1,
  prune = FALSE,
  lambda,
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
```

```
)

## S4 method for signature 'TwoStageBasket'
weights_cpp(design, n, n1, a = 1, b = 1, ...)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
a	first tuning parameter
b	second tuning parameter
prune	Whether baskets with a number of responses below the critical pooled value should be pruned before the final analysis. If this is TRUE then lambda is also required and if globalweight_fun is not NULL then globalweight_fun and globalweight_params are also used.
lambda	The posterior probability threshold. See details for more information.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
n1	The sample size per basket for the interim analysis in case of a two-stage design.

Details

weights_cpp calculates the weights based on an approach by Pan & Yuan (2017). The weight for two baskets i and j is found by at first calculating $S_{KS;i,j}$ as the Kolmogorov-Smirnov statistic, which is equal to the difference in response rates for binary variables. $S_{KS;i,j}$ is then transformed to $S_{i,j} = n^{1/4} S_{KS;i,j}$. Then the weight is found as $1/(1 + exp(a + b * log(S_{i,j})))$, where a and b are tuning parameters.

The function is generally not called by the user but passed to another function such as toer and pow to specificy how the weights are calculated.

Value

A matrix including the weights of all possible pairwise outcomes.

Methods (by class)

- weights_cpp(OneStageBasket): Calibrated power prior weights for a single-stage basket design.
- weights_cpp(TwoStageBasket): Calibrated power prior weights for a two-stage basket design.

References

Baumann, L., Sauer, L., & Kieser, M. (2024). A basket trial design based on power priors. arXiv:2309.06988.

Pan, H., Yuan, Y., & Xia, J. (2017). A calibrated power prior approach to borrow information from historical data with application to biosimilar clinical trials. Journal of the Royal Statistical Society Series C: Applied Statistics, 66(5), 979-996.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
toer(design, n = 15, lambda = 0.99, weight_fun = weights_cpp)
```

---

weights_fujikawa *Weights Based on Fujikawa et al.'s Design*

Description

Weights Based on Fujikawa et al.'s Design

Usage

```
weights_fujikawa(design, ...)

## S4 method for signature 'OneStageBasket'
weights_fujikawa(
  design,
  n,
  lambda,
  epsilon = 1.25,
  tau = 0.5,
  logbase = 2,
  prune = FALSE,
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)

## S4 method for signature 'TwoStageBasket'
weights_fujikawa(design, n, n1, epsilon = 1.25, tau = 0, logbase = 2, ...)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
lambda	The posterior probability threshold. See details for more information.

epsilon — A tuning parameter that determines the amount of borrowing. See details for more information.

tau — A tuning parameter that determines how similar the baskets have to be that borrowing occurs. See details for more information.

logbase — A tuning parameter that determines which logarithm base is used to compute the Jensen-Shannon divergence. See details for more information.

prune — Whether baskets with a number of responses below the critical pooled value should be pruned before the final analysis. If this is TRUE then lambda is also required and if globalweight_fun is not NULL then globalweight_fun and globalweight_params are also used.

globalweight_fun — Which function should be used to calculate the global weights.

globalweight_params — A list of tuning parameters specific to globalweight_fun.

n1 — The sample size per basket for the interim analysis in case of a two-stage design.

Details

weights_fujikawa calculates the weights used for sharing information between baskets based on the proposal by Fujikawa et al. (2020). The weight for two baskets i and j is found as $(1 - JSD(i,j))^{\varepsilon}$ where $JSD(i,j)$ is the Jensen-Shannon divergence between the individual posterior distributions of the response probabilities of basket i and j. Note that Fujikawa's weights also share the prior information between the baskets.

A small value of epsilon results in stronger borrowing also across baskets with heterogenous results. If epsilon is large then information is only borrowed between baskets with similar results. If a weight is smaller than tau it is set to 0, which results in no borrowing.

If prune = TRUE then the baskets with an observed number of baskets smaller than the pooled critical value are not borrowed from. The pooled critical value is the smallest integer c for which all null hypotheses can be rejected if the number of responses is exactly c for all baskets.

The function is generally not called by the user but passed to another function such as toer and pow to specificy how the weights are calculated.

Value

A matrix including the weights of all possible pairwise outcomes.

Methods (by class)

- weights_fujikawa(OneStageBasket): Fujikawa-weights for a single-stage basket design.
- weights_fujikawa(TwoStageBasket): Fujikawa-weights for a two-stage basket design.

References

Fujikawa, K., Teramukai, S., Yokota, I., & Daimon, T. (2020). A Bayesian basket trial design that borrows information across strata based on the similarity between the posterior distributions of the response probability. Biometrical Journal, 62(2), 330-338.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
toer(design, n = 15, lambda = 0.99, weight_fun = weights_fujikawa)
```

`weights_jsd` *Weights Based on the Jensen-Shannon Divergence*

Description

Weights Based on the Jensen-Shannon Divergence

Usage

```
weights_jsd(design, ...)

## S4 method for signature 'OneStageBasket'
weights_jsd(
  design,
  n,
  lambda,
  epsilon = 1.25,
  tau = 0.5,
  logbase = 2,
  prune = FALSE,
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)

## S4 method for signature 'TwoStageBasket'
weights_jsd(design, n, n1, epsilon = 1.25, tau = 0, logbase = 2, ...)
```

Arguments

`design`	An object of class `Basket` created by `setupOneStageBasket` or `setupTwoStageBasket`.
`...`	Further arguments.
`n`	The sample size per basket.
`lambda`	The posterior probability threshold. See details for more information.
`epsilon`	A tuning parameter that determines the amount of borrowing. See details for more information.
`tau`	A tuning parameter that determines how similar the baskets have to be that borrowing occurs. See details for more information.
`logbase`	A tuning parameter that determines which logarithm base is used to compute the Jensen-Shannon divergence. See details for more information.

prune
: Whether baskets with a number of responses below the critical pooled value should be pruned before the final analysis. If this is TRUE then lambda is also required and if globalweight_fun is not NULL then globalweight_fun and globalweight_params are also used.

globalweight_fun
: Which function should be used to calculate the global weights.

globalweight_params
: A list of tuning parameters specific to globalweight_fun.

n1
: The sample size per basket for the interim analysis in case of a two-stage design.

Details

weights_jsd calculates the weights used for sharing information between baskets based on the Jensen-Shannon divergence (JSD). The weight for two baskets i and j is found as $(1 - JSD(i, j))^{\varepsilon}$ where $JSD(i, j)$ is the Jensen-Shannon divergence between the individual posterior distributions of the response probabilities of basket i and j. This is identical to how the weights are calculated in weights_fujikawa, however when Fujikawa's weights are used the prior information is also shared.

A small value of epsilon results in stronger borrowing also across baskets with heterogenous results. If epsilon is large then information is only borrowed between baskets with similar results. If a weight is smaller than tau it is set to 0, which results in no borrowing.

If prune = TRUE then the baskets with an observed number of baskets smaller than the pooled critical value are not borrowed from. The pooled critical value is the smallest integer c for which all null hypotheses can be rejected if the number of responses is exactly c for all baskets.

The function is generally not called by the user but passed to another function such as toer and pow to specificy how the weights are calculated.

Value

A matrix including the weights of all possible pairwise outcomes.

Methods (by class)

- weights_jsd(OneStageBasket): Jensen-Shannon Divergence weights for a single-stage basket design.
- weights_jsd(TwoStageBasket): Jensen-Shannon Divergence weights for a two-stage basket design.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
toer(design, n = 15, lambda = 0.99, weight_fun = weights_jsd)
```

weights_mml *Weights Based on the Marginal Maximum Likelihood*

Description

Weights Based on the Marginal Maximum Likelihood

Usage

```
weights_mml(design, ...)

## S4 method for signature 'OneStageBasket'
weights_mml(
  design,
  n,
  prune = FALSE,
  lambda,
  globalweight_fun = NULL,
  globalweight_params = list(),
  ...
)

## S4 method for signature 'TwoStageBasket'
weights_mml(design, n, n1, ...)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
prune	Whether baskets with a number of responses below the critical pooled value should be pruned before the final analysis. If this is TRUE then lambda is also required and if globalweight_fun is not NULL then globalweight_fun and globalweight_params are also used.
lambda	The posterior probability threshold. See details for more information.
globalweight_fun	Which function should be used to calculate the global weights.
globalweight_params	A list of tuning parameters specific to globalweight_fun.
n1	The sample size per basket for the interim analysis in case of a two-stage design.

Details

`weights_mml` calculates the weights based on the marginal maximum likelihood approach by Gravestock & Held (2017). In this approach, the weight is found as the maximum of the marginal likelihood of the weight-parameter given the dataset that information should be borrowed from. However, since this can lead to non-symmetric weights (meaning that the amount of information that data set 1 borrows from data set 2 is generally not identical to the information data set 2 borrows from data set 1), a symmetrised version is used here: For the sharing-weight of Basket 1 and Basket 2 the MML is calculted two times - once conditional on the data of Basket 1 and once conditional on the data of Basket 2. The mean of these two weights is then used, resulting in symmetrical sharing.

Value

A matrix including the weights of all possible pairwise outcomes.

Methods (by class)

- `weights_mml(OneStageBasket)`: Maximum marginal likelihood weights for a single-stage basket design
- `weights_mml(TwoStageBasket)`: Maximum marginal likelihood weights for a two-stage basket design

References

Gravestock, I., & Held, L. (2017). Adaptive power priors with empirical Bayes for clinical trials. Pharmaceutical statistics, 16(5), 349-360.

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
toer(design, n = 15, lambda = 0.99, weight_fun = weights_mml)
```

---

weights_pool *Pooled Analysis*

Description

Pooled Analysis

Usage

```
weights_pool(design, ...)

## S4 method for signature 'OneStageBasket'
weights_pool(design, n, ...)

## S4 method for signature 'OneStageBasket'
weights_pool(design, n, ...)
```

```
## S4 method for signature 'TwoStageBasket'
weights_pool(design, n, n1, ...)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
n1	The sample size per basket for the interim analysis in case of a two-stage design.

Details

When weights_pool is used as a weight function, all data are pooled.

Value

A weight matrix where all weights are 1.

Methods (by class)

- weights_pool(OneStageBasket): Pooled analysis for a single-stage basket design
- weights_pool(OneStageBasket): Pooled analysis for a single-stage basket design
- weights_pool(TwoStageBasket): Pooled analysis for a two-stage basket design

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
toer(design, n = 15, lambda = 0.99, weight_fun = weights_pool)
```

---

weights_separate *Separate Analysis in Each Basket*

Description

Separate Analysis in Each Basket

Usage

```
weights_separate(design, ...)

## S4 method for signature 'OneStageBasket'
weights_separate(design, n, ...)

## S4 method for signature 'TwoStageBasket'
weights_separate(design, n, n1, ...)
```

Arguments

design	An object of class Basket created by setupOneStageBasket or setupTwoStageBasket.
...	Further arguments.
n	The sample size per basket.
n1	The sample size per basket for the interim analysis in case of a two-stage design.

Details

When weights_separate is used as a weight function, a separate analysis performed in each basket.

Value

A weight matrix where all weights are 0.

Methods (by class)

- weights_separate(OneStageBasket): Separate analysis for a single-stage basket design
- weights_separate(TwoStageBasket): Separate analysis for a two-stage basket design

Examples

```
design <- setupOneStageBasket(k = 3, p0 = 0.2)
toer(design, n = 15, lambda = 0.99, weight_fun = weights_separate)
```

Index

adjust_lambda, 2
adjust_lambda,OneStageBasket-method (adjust_lambda), 2
adjust_lambda,TwoStageBasket-method (adjust_lambda), 2

basket_test, 4
basket_test,OneStageBasket-method (basket_test), 4

check_mon_between, 6
check_mon_between,OneStageBasket-method (check_mon_between), 6
check_mon_within, 7
check_mon_within,OneStageBasket-method (check_mon_within), 7

ecd, 9
ecd,OneStageBasket-method (ecd), 9
ecd,TwoStageBasket-method (ecd), 9
ess, 11
ess,TwoStageBasket-method (ess), 11
estim, 12
estim,OneStageBasket-method (estim), 12
estim,TwoStageBasket-method (estim), 12

get_scenarios, 14, *21*
globalweights_diff, 15
globalweights_fix, 16

interim_posterior, 16
interim_posterior,TwoStageBasket-method (interim_posterior), 16
interim_postpred, 18
interim_postpred,TwoStageBasket-method (interim_postpred), 18

OneStageBasket, *24, 25*
OneStageBasket (OneStageBasket-class), 19
OneStageBasket-class, 19

opt_design, *14*, 20
opt_design,OneStageBasket-method (opt_design), 20
opt_design,TwoStageBasket-method (opt_design), 20

plot_weights, 22
plot_weights,OneStageBasket-method (plot_weights), 22
pow, *17*, *19*, 23, *30*, *32*, *34*
pow,OneStageBasket-method (pow), 23
pow,TwoStageBasket-method (pow), 23

setupOneStageBasket, 25
setupTwoStageBasket, 26

toer, *17*, *19*, 26, *30*, *32*, *34*
toer,OneStageBasket-method (toer), 26
toer,TwoStageBasket-method (toer), 26
TwoStageBasket, *26*
TwoStageBasket (TwoStageBasket-class), 28
TwoStageBasket-class, 28

weights_cpp, 29
weights_cpp,OneStageBasket-method (weights_cpp), 29
weights_cpp,TwoStageBasket-method (weights_cpp), 29
weights_fujikawa, 31, *34*
weights_fujikawa,OneStageBasket-method (weights_fujikawa), 31
weights_fujikawa,TwoStageBasket-method (weights_fujikawa), 31
weights_jsd, 33
weights_jsd,OneStageBasket-method (weights_jsd), 33
weights_jsd,TwoStageBasket-method (weights_jsd), 33
weights_mml, 35

weights_mml,OneStageBasket-method
(weights_mml), 35
weights_mml,TwoStageBasket-method
(weights_mml), 35
weights_pool, 36
weights_pool,OneStageBasket-method
(weights_pool), 36
weights_pool,TwoStageBasket-method
(weights_pool), 36
weights_separate, 37
weights_separate,OneStageBasket-method
(weights_separate), 37
weights_separate,TwoStageBasket-method
(weights_separate), 37