Package 'graphicalMCP'

August 27, 2026

Type Package

Title Graphical Multiple Comparison Procedures

Version 0.3.0

Description Multiple comparison procedures (MCPs) control the familywise error rate in clinical trials. Graphical MCPs include many commonly used procedures as special cases; see Bretz et al. (2011) <doi:10.1002/bimj.201000239>, Lu (2016) <doi:10.1002/sim.6985>, and Xi et al. (2017) <doi:10.1002/bimj.201600233>. This package is a low-dependency implementation of graphical MCPs which allow mixed types of tests. It also includes power simulations and visualization of graphical MCPs.

License Apache License (>= 2)

URL https://openpharma.github.io/graphicalMCP/, https://github.com/openpharma/graphicalMCP

BugReports https://github.com/openpharma/graphicalMCP/issues

Depends R (>= 4.1.0)

Imports matrixStats, mvtnorm

Suggests bench, dplyr, forcats, ggplot2, gMCP, gsDesign, gt, here, htmltools, igraph, knitr, lrstat (>= 0.3.3), prompt, rmarkdown, rpact, scales, testthat (>= 3.0.0), tibble, tictoc, tidyr, xfun

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

Language en-US

NeedsCompilation no

Author Dong Xi [aut, cre], Ethan Brockmann [aut], Gilead Sciences, Inc. [cph, fnd]

Maintainer Dong Xi <dong.xi1@gilead.com>

Repository CRAN

Date/Publication 2026-08-27 02:50:02 UTC

Contents

`adjust_p_bonferroni` *Calculate adjusted p-values*

Description

For an intersection hypothesis, an adjusted p-value is the smallest significance level at which the intersection hypothesis can be rejected. The intersection hypothesis can be rejected if its adjusted p-value is less than or equal to α. Currently, there are three test types supported:

- Bonferroni tests for `adjust_p_bonferroni()`,
- Parametric tests for `adjust_p_parametric()`,
  - Note that one-sided tests are required for parametric tests.
- Simes tests for `adjust_p_simes()`,
- Hochberg tests for `adjust_p_hochberg()`.

Usage

```
adjust_p_bonferroni(p, hypotheses)

adjust_p_parametric(
  p,
  hypotheses,
  test_corr = NULL,
  maxpts = 25000,
  abseps = 1e-06,
  releps = 0
)

adjust_p_simes(p, hypotheses)

adjust_p_hochberg(p, hypotheses)
```

Arguments

p	A numeric vector of p-values (unadjusted, raw), whose values should be between 0 & 1. The length should match the length of hypotheses.
hypotheses	A numeric vector of hypothesis weights. Must be a vector of values between 0 & 1 (inclusive). The length should match the length of p. The sum of hypothesis weights should not exceed 1.
test_corr	(Optional) A numeric matrix of correlations between test statistics, which is needed to perform parametric tests using adjust_p_parametric(). The number of rows and columns of this correlation matrix should match the length of p.
maxpts	(Optional) An integer scalar for the maximum number of function values, which is needed to perform parametric tests using the mvtnorm::GenzBretz algorithm. The default is 25000.
abseps	(Optional) A numeric scalar for the absolute error tolerance, which is needed to perform parametric tests using the mvtnorm::GenzBretz algorithm. The default is 1e-6.
releps	(Optional) A numeric scalar for the relative error tolerance as double, which is needed to perform parametric tests using the mvtnorm::GenzBretz algorithm. The default is 0.

Value

A single adjusted p-value for the intersection hypothesis.

References

Bretz, F., Maurer, W., Brannath, W., and Posch, M. (2009). A graphical approach to sequentially rejective multiple test procedures. *Statistics in Medicine*, 28(4), 586-604.

Lu, K. (2016). Graphical approaches using a Bonferroni mixture of weighted Simes tests. *Statistics in Medicine*, 35(22), 4041-4055.

Xi, D., Glimm, E., Maurer, W., and Bretz, F. (2017). A unified framework for weighted parametric multiple test procedures. *Biometrical Journal*, 59(5), 918-931.

Xi, D., and Bretz, F. (2019). Symmetric graphs for equally weighted tests, with application to the Hochberg procedure. *Statistics in Medicine*, 38(27), 5268-5282.

See Also

`adjust_weights_parametric()` for adjusted hypothesis weights using parametric tests, `adjust_weights_simes()` for adjusted hypothesis weights using Simes tests, `adjust_weights_hochberg()` for adjusted hypothesis weights using Hochberg tests.

Examples

```
hypotheses <- c(H1 = 0.5, H2 = 0.25, H3 = 0.25)
p <- c(0.019, 0.025, 0.05)
adjust_p_bonferroni(p, hypotheses)
set.seed(1234)
hypotheses <- c(H1 = 0.5, H2 = 0.25, H3 = 0.25)
p <- c(0.019, 0.025, 0.05)
# Using the `mvtnorm::GenzBretz` algorithm
corr <- matrix(0.5, nrow = 3, ncol = 3)
diag(corr) <- 1
adjust_p_parametric(p, hypotheses, corr)
hypotheses <- c(H1 = 0.5, H2 = 0.25, H3 = 0.25)
p <- c(0.019, 0.025, 0.05)
adjust_p_simes(p, hypotheses)
hypotheses <- c(H1 = .25, H2 = .25, H3 = 0.25, H4 = 0.25)
p <- c(0.019, 0.025, 0.05, .05)
adjust_p_hochberg(p, hypotheses)
```

`adjust_weights_parametric`

Calculate adjusted hypothesis weights

Description

An intersection hypothesis can be rejected if its p-values are less than or equal to their adjusted significance levels, which are their adjusted hypothesis weights times α. For Bonferroni tests, their adjusted hypothesis weights are their hypothesis weights of the intersection hypothesis. Additional adjustment is needed for parametric, Simes, and Hochberg tests:

- Parametric tests for `adjust_weights_parametric()`,
  - Note that one-sided tests are required for parametric tests.
- Simes tests for `adjust_weights_simes()`,
- Hochberg tests for `adjust_weights_hochberg()`.

Usage

```
adjust_weights_parametric(
  matrix_weights,
  matrix_intersections,
  test_corr,
  alpha,
  test_groups,
  ...
)

adjust_weights_simes(matrix_weights, p, test_groups)

adjust_weights_hochberg(matrix_weights, matrix_intersections, p, test_groups)
```

Arguments

`matrix_weights`	(Optional) A matrix of hypothesis weights of all intersection hypotheses. This can be obtained as the second half of columns from the output of `graph_generate_weights()`.
`matrix_intersections`	(Optional) A matrix of hypothesis indicators of all intersection hypotheses. This can be obtained as the first half of columns from the output of `graph_generate_weights()`.
`test_corr`	(Optional) A numeric matrix of correlations between test statistics, which is needed to perform parametric tests using `adjust_weights_parametric()`. The number of rows and columns of this correlation matrix should match the length of `p`.
`alpha`	(Optional) A numeric value of the overall significance level, which should be between 0 & 1. The default is 0.025 for one-sided hypothesis testing problems; another common choice is 0.05 for two-sided hypothesis testing problems. Note when parametric tests are used, only one-sided tests are supported.
`test_groups`	(Optional) A list of numeric vectors specifying hypotheses to test together. Grouping is needed to correctly perform Simes and parametric tests.
`...`	Additional arguments to perform parametric tests using the `mvtnorm::GenzBretz` algorithm. `maxpts` is an integer scalar for the maximum number of function values, whose default value is 25000. `abseps` is a numeric scalar for the absolute error tolerance, whose default value is 1e-6. `releps` is a numeric scalar for the relative error tolerance as double, whose default value is 0.
`p`	(Optional) A numeric vector of p-values (unadjusted, raw), whose values should be between 0 & 1. The length should match the number of columns of `matrix_weights`.

Value

- `adjust_weights_parametric()` returns a matrix with the same dimensions as `matrix_weights`, whose hypothesis weights have been adjusted according to parametric tests.
- `adjust_weights_simes()` returns a matrix with the same dimensions as `matrix_weights`, whose hypothesis weights have been adjusted according to Simes tests.
- `adjust_weights_hochberg()` returns a matrix with the same dimensions as `matrix_weights`, whose hypothesis weights have been adjusted according to Hochberg tests.

References

Lu, K. (2016). Graphical approaches using a Bonferroni mixture of weighted Simes tests. *Statistics in Medicine*, 35(22), 4041-4055.

Xi, D., Glimm, E., Maurer, W., and Bretz, F. (2017). A unified framework for weighted parametric multiple test procedures. *Biometrical Journal*, 59(5), 918-931.

Xi, D., and Bretz, F. (2019). Symmetric graphs for equally weighted tests, with application to the Hochberg procedure. *Statistics in Medicine*, 38(27), 5268-5282.

See Also

`adjust_p_parametric()` for adjusted p-values using parametric tests, `adjust_p_simes()` for adjusted p-values using Simes tests, `adjust_p_hochberg()` for adjusted p-values using Hochberg tests.

Examples

```
alpha <- 0.025
num_hyps <- 4
g <- bonferroni_holm(num_hyps)
weighting_strategy <- graph_generate_weights(g)
matrix_intersections <- weighting_strategy[, seq_len(num_hyps)]
matrix_weights <- weighting_strategy[, -seq_len(num_hyps)]

set.seed(1234)
adjust_weights_parametric(
  matrix_weights = matrix_weights,
  matrix_intersections = matrix_intersections,
  test_corr = list(diag(2), diag(2)),
  alpha = alpha,
  test_groups = list(1:2, 3:4)
)
alpha <- 0.025
p <- c(0.018, 0.01, 0.105, 0.006)
num_hyps <- length(p)
g <- bonferroni_holm(num_hyps)
weighting_strategy <- graph_generate_weights(g)
matrix_intersections <- weighting_strategy[, seq_len(num_hyps)]
matrix_weights <- weighting_strategy[, -seq_len(num_hyps)]

adjust_weights_simes(
  matrix_weights = matrix_weights,
  p = p,
  test_groups = list(1:2, 3:4)
)
alpha <- 0.025
p <- c(0.018, 0.01, 0.105, 0.006)
num_hyps <- length(p)
g <- bonferroni_holm(num_hyps)
weighting_strategy <- graph_generate_weights(g)
matrix_intersections <- weighting_strategy[, seq_len(num_hyps)]
matrix_weights <- weighting_strategy[, -seq_len(num_hyps)]
```

```
adjust_weights_hochberg(
  matrix_weights = matrix_weights,
  matrix_intersections = matrix_intersections,
  p = p,
  test_groups = list(1:2, 3:4)
)
```

---

as_initial_graph *Convert between graphicalMCP, gMCP, and igraph graph classes*

Description

Graph objects have different structures and attributes in graphicalMCP, gMCP, and igraph R packages. These functions convert between different classes to increase compatibility.

Note that igraph and gMCP have additional attributes for vertices, edges, or a graph itself. These conversion functions only handle attributes related to hypothesis names, hypothesis weights and transition weights. Other attributes will be dropped when converting.

Usage

```
as_initial_graph(graph)

## S3 method for class 'graphMCP'
as_initial_graph(graph)

## S3 method for class 'igraph'
as_initial_graph(graph)

as_graphMCP(graph)

## S3 method for class 'initial_graph'
as_graphMCP(graph)

as_igraph(graph)

## S3 method for class 'initial_graph'
as_igraph(graph)
```

Arguments

graph	An initial_graph object from the graphicalMCP package, a graphMCP object from the gMCP package, or an igraph object from the igraph package, depending on the conversion type.

Value

- `as_graphMCP()` returns a `graphMCP` object for the `gMCP` package.
- `as_igraph()` returns an `igraph` object for the `igraph` package.
- `as_initial_graph()` returns an `initial_graph` object for the `graphicalMCP` package.

References

Csardi, G., Nepusz, T., Traag, V., Horvat, S., Zanini, F., Noom, D., and Mueller, K. (2024). *igraph*: Network analysis and visualization in R. R package version 2.0.3. https://CRAN.R-project.org/package=igraph.

Rohmeyer, K., and Klinglmueller, K. (2024). *gMCP*: Graph based multiple test procedures. R package version 0.8-17. https://cran.r-project.org/package=gMCP.

See Also

`graph_create()` for the initial graph used in the `graphicalMCP` package.

Examples

```
g_graphicalMCP <- random_graph(5)

if (requireNamespace("gMCP", quietly = TRUE)) {
  g_gMCP <- as_graphMCP(g_graphicalMCP)

  all.equal(g_graphicalMCP, as_initial_graph(g_gMCP))
}

if (requireNamespace("igraph", quietly = TRUE)) {
  g_igraph <- as_igraph(g_graphicalMCP)

  all.equal(g_graphicalMCP, as_initial_graph(g_igraph))
}
```

---

bonferroni *Example graphs of commonly used multiple comparison procedures*

Description

Built-in functions to quickly generate select graphical multiple comparison procedures.

Usage

```
bonferroni(num_hyps, hyp_names = NULL)

bonferroni_weighted(hypotheses, hyp_names = NULL)

bonferroni_holm(num_hyps, hyp_names = NULL)
```

```
bonferroni_holm_weighted(hypotheses, hyp_names = NULL)

dunnett_single_step(num_hyps, hyp_names = NULL)

dunnett_single_step_weighted(hypotheses, hyp_names = NULL)

dunnett_closure_weighted(hypotheses, hyp_names = NULL)

hochberg(num_hyps, hyp_names = NULL)

hommel(num_hyps, hyp_names = NULL)

huque_etal(hyp_names = NULL)

fallback(hypotheses, hyp_names = NULL)

fallback_improved_1(hypotheses, hyp_names = NULL)

fallback_improved_2(hypotheses, epsilon = 1e-04, hyp_names = NULL)

fixed_sequence(num_hyps, hyp_names = NULL)

sidak(num_hyps, hyp_names = NULL)

simple_successive_1(hyp_names = NULL)

simple_successive_2(hyp_names = NULL)

two_doses_two_primary_two_secondary(hyp_names = NULL)

three_doses_two_primary_two_secondary(hyp_names = NULL)

random_graph(num_hyps, hyp_names = NULL)
```

Arguments

num_hyps	(Optional) Number of hypotheses in a graphical multiple comparison procedure.
hyp_names	(Optional) A character vector of hypothesis names. The length should match num_hyps and the length of hypotheses. If hyp_names are not specified, hypotheses will be named sequentially as H1, H2, ...
hypotheses	(Optional) A numeric vector of hypothesis weights in a graphical multiple comparison procedure. Must be a vector of values between 0 & 1 (inclusive). The length should match num_hyps and the length of hyp_names. The sum of hypothesis weights should not exceed 1.
epsilon	(Optional) A numeric scalar indicating the value of the ϵ edge. This should be a much smaller value than hypothesis and transition weights. The default is 1e-4.

Value

An S3 object as returned by `graph_create()`.

References

Bretz, F., Maurer, W., Brannath, W., and Posch, M. (2009). A graphical approach to sequentially rejective multiple test procedures. *Statistics in Medicine*, 28(4), 586-604.

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

Hochberg, Y. (1988). A sharper Bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4), 800-802.

Hommel, G. (1988). A stagewise rejective multiple test procedure based on a modified Bonferroni test. *Biometrika*, 75(2), 383-386.

Huque, M. F., Alosh, M., and Bhore, R. (2011). Addressing multiplicity issues of a composite endpoint and its components in clinical trials. *Journal of Biopharmaceutical Statistics*, 21(4), 610-634.

Maurer, W., Hothorn, L., and Lehmacher, W. (1995). Multiple comparisons in drug clinical trials and preclinical assays: a-priori ordered hypotheses. *Biometrie in der chemisch-pharmazeutischen Industrie*, 6, 3-18.

Šidák, Z. (1967). Rectangular confidence regions for the means of multivariate normal distributions. *Journal of the American Statistical Association*, 62(318), 626-633.

Westfall, P. H., and Krishen, A. (2001). Optimally weighted, fixed sequence and gatekeeper multiple testing procedures. *Journal of Statistical Planning and Inference*, 99(1), 25-40.

Wiens, B. L. (2003). A fixed sequence Bonferroni procedure for testing multiple endpoints. *Pharmaceutical Statistics*, 2(3), 211-215.

Wiens, B. L., and Dmitrienko, A. (2005). The fallback procedure for evaluating a single family of hypotheses. *Journal of Biopharmaceutical Statistics*, 15(6), 929-942.

Xi, D., and Bretz, F. (2019). Symmetric graphs for equally weighted tests, with application to the Hochberg procedure. *Statistics in Medicine*, 38(27), 5268-5282.

See Also

`graph_create()` for a general way to create the initial graph.

Examples

```
# Bretz et al. (2009)
bonferroni(num_hyps = 3)
# Bretz et al. (2009)
hypotheses <- c(0.5, 0.3, 0.2)
bonferroni_weighted(hypotheses)
# Bretz et al. (2009)
bonferroni_holm(num_hyps = 3)
# Bretz et al. (2009)
hypotheses <- c(0.5, 0.3, 0.2)
```

```
bonferroni_holm_weighted(hypotheses)
# Xi et al. (2017)
dunnett_single_step(num_hyps = 3)
# Xi et al. (2017)
hypotheses <- c(0.5, 0.3, 0.2)
dunnett_single_step_weighted(hypotheses)
# Xi et al. (2009)
hypotheses <- c(0.5, 0.3, 0.2)
dunnett_closure_weighted(hypotheses)
# Hochberg (1988)
hochberg(num_hyps = 3)
# Hommel (1988)
hommel(num_hyps = 3)
# Huque et al. (2011)
huque_etal()
# Wiens (2003)
hypotheses <- c(0.5, 0.3, 0.2)
fallback(hypotheses)
# Wiens and Dmitrienko (2005)
hypotheses <- c(0.5, 0.3, 0.2)
fallback_improved_1(hypotheses)
# Bretz et al. (2009)
hypotheses <- c(0.5, 0.3, 0.2)
fallback_improved_2(hypotheses)
# Maurer et al. (1995); Westfall and Krishen (2001)
fixed_sequence(num_hyps = 3)
# sidak (1967)
sidak(num_hyps = 3)
# Figure 1 in Bretz et al. (2011)
simple_successive_1()
# Figure 4 in Bretz et al. (2011)
simple_successive_2()
# Figure 6 in Xi and Bretz et al. (2019)
two_doses_two_primary_two_secondary()
# Add another dose to Figure 6 in Xi and Bretz et al. (2019)
three_doses_two_primary_two_secondary()
# Create a random graph with three hypotheses
random_graph(num_hyps = 3)
```

`graph_calculate_power` *Calculate power values for a graphical multiple comparison procedure*

Description

Under the alternative hypotheses, the distribution of test statistics is assumed to be a multivariate normal distribution. Given this distribution, this function calculates power values for a graphical multiple comparison procedure. By default, it calculate the local power, which is the probability to reject an individual hypothesis, the probability to reject at least one hypothesis, the probability to reject all hypotheses, the expected number of rejections, and the probability of user-defined

success criteria. See `vignette("shortcut-testing")` and `vignette("closed-testing")` for more illustration of power calculation.

Usage

```
graph_calculate_power(
  graph,
  alpha = 0.025,
  power_marginal = rep(alpha, length(graph$hypotheses)),
  test_groups = list(seq_along(graph$hypotheses)),
  test_types = c("bonferroni"),
  test_corr = rep(list(NA), length(test_types)),
  sim_n = 1e+05,
  sim_corr = diag(length(graph$hypotheses)),
  sim_success = NULL,
  verbose = FALSE
)
```

Arguments

`graph`	An initial graph as returned by `graph_create()`.
`alpha`	A numeric value of the one-sided overall significance level, which should be between 0 & 1. The default is 0.025 for one-sided hypothesis testing. Note that only one-sided tests are supported.
`power_marginal`	A numeric vector of marginal power values to use when simulating p-values. See Details for more on the simulation process.
`test_groups`	A list of numeric vectors specifying hypotheses to test together. Grouping is needed to correctly perform Simes and parametric tests.
`test_types`	A character vector of test types to apply to each test group. This is needed to correctly perform Simes and parametric tests. The length should match the number of elements in `test_groups`.
`test_corr`	(Optional) A list of numeric correlation matrices. Each entry in the list should correspond to each test group. For a test group using Bonferroni or Simes tests, its corresponding entry in `test_corr` should be `NA`. For a test group using parametric tests, its corresponding entry in `test_corr` should be a numeric correlation matrix specifying the correlation between test statistics for hypotheses in this test group. The length should match the number of elements in `test_groups`.
`sim_n`	An integer scalar specifying the number of simulations. The default is 1e5.
`sim_corr`	A numeric matrix of correlations between test statistics for all hypotheses. The dimensions should match the number of hypotheses in `graph`.
`sim_success`	A list of user-defined functions to specify the success criteria. Functions must take one simulation's logical vector of results as an input, and return a length-one logical vector. For instance, if "success" means rejecting hypotheses 1 and 2, use `sim_success = list("1 and 2" = function(x) x[1] && x[2])`. If the list is not named, the function body will be used as the name. Lambda functions also work starting with R 4.1, e.g. `sim_success = list(\(x) x[3] || x[4])`.

verbose A logical scalar specifying whether the details of power simulations should be included in results. The default is verbose = FALSE.

Value

A power_report object with a list of 3 elements:

- inputs - Input parameters, which is a list of:
  - graph - Initial graph,
  - alpha - Overall significance level,
  - test_groups - Groups of hypotheses for different types of tests,
  - test_types - Different types of tests,
  - test_corr - Correlation matrices for parametric tests,
  - sim_n - Number of simulations,
  - power_marginal - Marginal power of all hypotheses
  - sim_corr - Correlation matrices for simulations,
  - sim_success - User-defined success criteria.
- power - A list of power values
  - power_local - Local power of all hypotheses, which is the proportion of simulations in which each hypothesis is rejected,
  - rejection_expected - Expected (average) number of rejected hypotheses,
  - power_at_least_1 - Power to reject at least one hypothesis,
  - power_all - Power to reject all hypotheses,
  - power_success - Power of user-defined success, which is the proportion of simulations in which the user-defined success criterion
  - sim_success is met.
- details - An optional list of datasets showing simulated p-values and results for each simulation.

Simulation details

The power calculation is based on simulations. The distribution to simulate from is determined as a multivariate normal distribution by power_marginal and sim_corr. In particular, power_marginal is a vector of marginal power values for all hypotheses. The marginal power is the power to reject the null hypothesis at the significance level alpha *without multiplicity adjustment*. This value could be readily available from standard software and other R packages. Then we can determine the mean of the multivariate normal distribution as

$$\Phi^{-1}(1-\alpha) - \Phi^{-1}(1-d_i)$$

, which is often called the non-centrality parameter or the drift parameter. Here d_i is the marginal power power_marginal of hypothesis i. Given the correlation matrix sim_corr, we can simulate from this multivariate normal distribution using the mvtnorm R package (Genz and Bretz, 2009).

Each set simulated values can be used to calculate the corresponding one-sided p-values. Then this set of p-values are plugged into the graphical multiple comparison procedure to determine which hypotheses are rejected. This process is repeated n_sim times to produce the power values as the proportion of simulations in which a particular success criterion is met.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011a). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

Bretz, F., Maurer, W., and Hommel, G. (2011b). Test and power considerations for multiple endpoint analyses using sequentially rejective graphical procedures. *Statistics in Medicine*, 30(13), 1489-1501.

Genz, A., and Bretz, F. (2009). *Computation of Multivariate Normal and t Probabilities*, series Lecture Notes in Statistics. Springer-Verlag, Heidelberg.

Lu, K. (2016). Graphical approaches using a Bonferroni mixture of weighted Simes tests. *Statistics in Medicine*, 35(22), 4041-4055.

Xi, D., Glimm, E., Maurer, W., and Bretz, F. (2017). A unified framework for weighted parametric multiple test procedures. *Biometrical Journal*, 59(5), 918-931.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 4 in Bretz et al. (2011a).
alpha <- 0.025
hypotheses <- c(0.5, 0.5, 0, 0)
delta <- 0.5
transitions <- rbind(
  c(0, delta, 1 - delta, 0),
  c(delta, 0, 0, 1 - delta),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

marginal_power <- c(0.8, 0.8, 0.7, 0.9)
corr1 <- matrix(0.5, nrow = 2, ncol = 2)
diag(corr1) <- 1
corr <- rbind(
  cbind(corr1, 0.5 * corr1),
  cbind(0.5 * corr1, corr1)
)
success_fns <- list(
  # Probability to reject both H1 and H2
  `H1andH2` = function(x) x[1] & x[2],
  # Probability to reject both (H1 and H3) or (H2 and H4)
  `(H1andH3)or(H2andH4)` = function(x) (x[1] & x[3]) | (x[2] & x[4])
)
set.seed(1234)
# Bonferroni tests
# Reduce the number of simulations to save time for package compilation
power_output <- graph_calculate_power(
  g,
  alpha,
  sim_corr = corr,
```

```
  sim_n = 1e2,
  power_marginal = marginal_power,
  sim_success = success_fns
)

# Parametric tests for H1 and H2; Simes tests for H3 and H4
# User-defined success: to reject H1 or H2; to reject H1 and H2
# Reduce the number of simulations to save time for package compilation
graph_calculate_power(
  g,
  alpha,
  test_groups = list(1:2, 3:4),
  test_types = c("parametric", "simes"),
  test_corr = list(corr1, NA),
  sim_n = 1e2,
  sim_success = list(
    function(.) .[1] || .[2],
    function(.) .[1] && .[2]
  )
)
```

`graph_create` *Create the initial graph for a multiple comparison procedure*

Description

A graphical multiple comparison procedure is represented by 1) a vector of initial hypothesis weights `hypotheses`, and 2) a matrix of initial transition weights `transitions`. This function creates the initial graph object using hypothesis weights and transition weights.

Usage

```
graph_create(hypotheses, transitions, hyp_names = NULL)
```

Arguments

`hypotheses`	A numeric vector of hypothesis weights in a graphical multiple comparison procedure. Must be a vector of values between 0 & 1 (inclusive). The length should match the row and column lengths of `transitions`. The sum of hypothesis weights should not exceed 1.
`transitions`	A numeric matrix of transition weights between hypotheses in a graphical multiple comparison procedure. Must be a square matrix of values between 0 & 1 (inclusive). The row and column lengths should match the length of `hypotheses`. Each row (Transition weights leaving a hypothesis) can sum to no more than 1. The diagonal entries (Transition weights from a hypothesis to itself) must be all 0s.
`hyp_names`	(Optional) A character vector of hypothesis names. If not provided, names from `hypotheses` and `transitions` will be used. If names are not specified, hypotheses will be named sequentially as H1, H2, ...

Value

An S3 object of class `initial_graph` with a list of 2 elements:

- Hypothesis weights `hypotheses`.
- Transition weights `transitions`.

Validation of inputs

Inputs are also validated to make sure of the validity of the graph:

- Hypothesis weights `hypotheses` are numeric.
- Transition weights `transitions` are numeric.
- Length of `hypotheses` and dimensions of `transitions` match.
- Hypothesis weights `hypotheses` must be non-negative and sum to no more than 1.
- Transition weights `transitions`:
  - Values must be non-negative.
  - Rows must sum to no more than 1.
  - Diagonal entries must be all 0.
- Hypothesis names `hyp_names` override names in `hypotheses` or `transitions`.

References

Bretz, F., Maurer, W., Brannath, W., and Posch, M. (2009). A graphical approach to sequentially rejective multiple test procedures. *Statistics in Medicine*, 28(4), 586-604.

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

`graph_update()` for the updated graph after hypotheses being deleted from the initial graph.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
hyp_names <- c("H11", "H12", "H21", "H22")
g <- graph_create(hypotheses, transitions, hyp_names)
g
```

```
# Explicit names override names in `hypotheses` (with a warning)
hypotheses <- c(h1 = 0.5, h2 = 0.5, h3 = 0, h4 = 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions, hyp_names)
g

# Use names in `transitions`
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  H1 = c(0, 0, 1, 0),
  H2 = c(0, 0, 0, 1),
  H3 = c(0, 1, 0, 0),
  H4 = c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)
g

# Unmatched names in `hypotheses` and `transitions` (with an error)
hypotheses <- c(h1 = 0.5, h2 = 0.5, h3 = 0, h4 = 0)
transitions <- rbind(
  H1 = c(0, 0, 1, 0),
  H2 = c(0, 0, 0, 1),
  H3 = c(0, 1, 0, 0),
  H4 = c(1, 0, 0, 0)
)
try(
  g <- graph_create(hypotheses, transitions)
)

# When names are not specified, hypotheses are numbered sequentially as
# H1, H2, ...
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)
g
```

---

graph_generate_weights

Generate the weighting strategy based on a graphical multiple comparison procedure

---

Description

A graphical multiple comparison procedure defines a closed test procedure, which tests each intersection hypothesis and reject an individual hypothesis if all intersection hypotheses involving it have been rejected. An intersection hypothesis represents the parameter space where individual null hypotheses involved are true simultaneously.

The closure based on a graph consists of all updated graphs (corresponding to intersection hypotheses) after all combinations of hypotheses are deleted. For a graphical multiple comparison procedure with m hypotheses, there are $2^m - 1$ updated graphs (intersection hypotheses), including the initial graph (the overall intersection hypothesis). The weighting strategy of this graph consists of hypothesis weights from all $2^m - 1$ updated graphs (intersection hypotheses). The algorithm to derive the weighting strategy is based on Algorithm 1 in Bretz et al. (2011).

Usage

```
graph_generate_weights(graph)
```

Arguments

graph	An initial graph as returned by `graph_create()`.

Value

A numeric matrix of all intersection hypotheses and their hypothesis weights. For a graphical multiple comparison procedure with m hypotheses, the number of rows is $2^m - 1$, each of which corresponds to an intersection hypothesis. The number of columns is $2 \cdot m$. The first m columns indicate which individual hypotheses are included in a given intersection hypothesis and the second half of columns provide hypothesis weights for each individual hypothesis for a given intersection hypothesis.

Performance

Generation of intersection hypotheses is closely related to the power set of a given set of indices. As the number of hypotheses increases, the memory and time usage can grow quickly (e.g., at a rate of $O(2^n)$). There are also multiple ways to implement Algorithm 1 in Bretz et al. (2011). See `vignette("generate-closure")` for more information about generating intersection hypotheses and comparisons of different approaches to calculate weighting strategies.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

`graph_test_closure()` for graphical multiple comparison procedures using the closed test.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

graph_generate_weights(g)
```

---

graph_rejection_orderings

Find alternate rejection orderings (sequences) for shortcut tests

---

Description

When multiple hypotheses are rejected by using `graph_test_shortcut()`, there may be multiple orderings or sequences in which hypotheses are rejected one by one. The default order in `graph_test_shortcut()` is based on the adjusted p-values, from the smallest to the largest. This function `graph_rejection_orderings()` provides all possible and valid orders (or sequences) of rejections. Although the order of rejection does not affect the final rejection decisions Bretz et al. (2009), different sequences could offer different ways to explain the step-by-step process of shortcut graphical multiple comparison procedures.

Usage

```
graph_rejection_orderings(shortcut_test_result)
```

Arguments

`shortcut_test_result`
: A `graph_report` object as returned by `graph_test_shortcut()`.

Value

A modified `graph_report` object containing all valid orderings of rejections of hypotheses

References

Bretz, F., Maurer, W., Brannath, W., and Posch, M. (2009). A graphical approach to sequentially rejective multiple test procedures. *Statistics in Medicine*, 28(4), 586-604.

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

graph_test_shortcut() for shortcut graphical multiple comparison procedures.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 4 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
delta <- 0.5
transitions <- rbind(
  c(0, delta, 1 - delta, 0),
  c(delta, 0, 0, 1 - delta),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

p <- c(0.018, 0.01, 0.105, 0.006)
alpha <- 0.025

shortcut_testing <- graph_test_shortcut(g, p, alpha, verbose = TRUE)

# Reject H1, H2, and H4
shortcut_testing$outputs$rejected

# Default order of rejections: H2, H1, H4
shortcut_testing$details$del_seq

# There is another valid sequence of rejection: H2, H4, H1
graph_rejection_orderings(shortcut_testing)$valid_orderings

# Finally, intermediate updated graphs can be obtained by providing the order
# of rejections into `[graph_update()]`
graph_update(g, delete = c(2, 4, 1))
```

---

graph_test_closure *Perform closed graphical multiple comparison procedures*

Description

Closed graphical multiple comparison procedures, or graphical multiple comparison procedures based on the closure, generate the closure based on a graph consisting of all intersection hypotheses. It tests each intersection hypothesis and rejects an individual hypothesis if all intersection hypotheses involving it have been rejected. An intersection hypothesis represents the parameter space where individual null hypotheses involved are true simultaneously.

For a graphical multiple comparison procedure with m hypotheses, there are $2^m - 1$ intersection hypotheses. For each intersection hypothesis, a test type could be chosen to determine how to reject the intersection hypothesis. Current choices of test types include Bonferroni, Simes and parametric.

This implementation offers a more general framework covering Bretz et al. (2011), Lu (2016), and Xi et al. (2017). See `vignette("closed-testing")` for more illustration of closed test procedures and interpretation of their outputs.

Usage

```
graph_test_closure(
  graph,
  p,
  alpha = 0.025,
  test_groups = list(seq_along(graph$hypotheses)),
  test_types = c("bonferroni"),
  test_corr = rep(list(NA), length(test_types)),
  verbose = FALSE,
  test_values = FALSE
)
```

Arguments

`graph`	An initial graph as returned by `graph_create()`.
`p`	A numeric vector of p-values (unadjusted, raw), whose values should be between 0 & 1. The length should match the number of hypotheses in `graph`.
`alpha`	A numeric value of the overall significance level, which should be between 0 & 1. The default is 0.025 for one-sided hypothesis testing problems; another common choice is 0.05 for two-sided hypothesis testing problems. Note when parametric tests are used, only one-sided tests are supported.
`test_groups`	A list of numeric vectors specifying hypotheses to test together. Grouping is needed to correctly perform Simes and parametric tests.
`test_types`	A character vector of test types to apply to each test group. This is needed to correctly perform Simes and parametric tests. The length should match the number of elements in `test_groups`.
`test_corr`	(Optional) A list of numeric correlation matrices. Each entry in the list should correspond to each test group. For a test group using Bonferroni or Simes tests, its corresponding entry in `test_corr` should be `NA`. For a test group using parametric tests, its corresponding entry in `test_corr` should be a numeric correlation matrix specifying the correlation between test statistics for hypotheses in this test group. The length should match the number of elements in `test_groups`.
`verbose`	A logical scalar specifying whether the details of the adjusted p-value calculations should be included in results. When `verbose = TRUE`, adjusted p-values are provided for each intersection hypothesis. The default is `verbose = FALSE`.
`test_values`	A logical scalar specifying whether adjusted significance levels should be provided for each hypothesis. When `test_values = TRUE`, it provides an equivalent way of performing graphical multiple comparison procedures by comparing each p-value with its significance level. If the p-value of a hypothesis is less than or equal to its significance level, the hypothesis is rejected. The default is `test_values = FALSE`.

Value

A `graph_report` object with a list of 4 elements:

- `inputs` - Input parameters, which is a list of:
  - `graph` - Initial graph,
  - `p` - (Unadjusted or raw) p-values,
  - `alpha` - Overall significance level,
  - `test_groups` - Groups of hypotheses for different types of tests,
  - `test_types` - Different types of tests,
  - `test_corr` - Correlation matrices for parametric tests.
- `outputs` - Output parameters, which is a list of:
  - `adjusted_p` - Adjusted p-values,
  - `rejected` - Rejected hypotheses,
  - `graph` - Updated graph after deleting all rejected hypotheses.
- `details` - Verbose outputs with adjusted p-values for intersection hypotheses, if `verbose = TRUE`.
- `test_values` - Adjusted significance levels, if `test_values = TRUE`.

Details for test specification

Test specification includes three components: `test_groups`, `test_types`, and `test_corr`. Alignment among entries in these components is important for correct implementation. There are two ways to provide test specification. The first approach is the "unnamed" approach, which assumes that all 3 components are ordered the same way, i.e., the n-th element of `test_types` and `test_corr` should apply to the n-th group in `test_groups`. The second "named" approach uses the name of each element of each component to connect the element of `test_types` and `test_corr` with the correct element of `test_groups`. Consistency should be ensured for correct implementation.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

Lu, K. (2016). Graphical approaches using a Bonferroni mixture of weighted Simes tests. *Statistics in Medicine*, 35(22), 4041-4055.

Xi, D., Glimm, E., Maurer, W., and Bretz, F. (2017). A unified framework for weighted parametric multiple test procedures. *Biometrical Journal*, 59(5), 918-931.

See Also

`graph_test_shortcut()` for shortcut graphical multiple comparison procedures.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses
# (H1 and H2) and two secondary hypotheses (H3 and H4)
# See Figure 4 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
delta <- 0.5
transitions <- rbind(
  c(0, delta, 1 - delta, 0),
  c(delta, 0, 0, 1 - delta),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

p <- c(0.018, 0.01, 0.105, 0.006)
alpha <- 0.025

# Closed graphical multiple comparison procedure using Bonferroni tests
# Same results as `graph_test_shortcut(g, p, alpha)`
graph_test_closure(g, p, alpha)

# Closed graphical multiple comparison procedure using parametric tests for
# H1 and H2, and Bonferroni tests for H3 and H4
set.seed(1234)
corr_list <- list(matrix(c(1, 0.5, 0.5, 1), nrow = 2), NA)
graph_test_closure(
  graph = g,
  p = p,
  alpha = alpha,
  test_groups = list(1:2, 3:4),
  test_types = c("parametric", "bonferroni"),
  test_corr = corr_list
)
# The "named" approach to obtain the same results
# Note that "group2" appears before "group1" in `test_groups`
set.seed(1234)
corr_list <- list(group1 = matrix(c(1, 0.5, 0.5, 1), nrow = 2), group2 = NA)
graph_test_closure(
  graph = g,
  p = p,
  alpha = alpha,
  test_groups = list(group1 = 1:2, group2 = 3:4),
  test_types = c(group2 = "bonferroni", group1 = "parametric"),
  test_corr = corr_list
)

# Closed graphical multiple comparison procedure using parametric tests for
# H1 and H2, and Simes tests for H3 and H4
set.seed(1234)
graph_test_closure(
  graph = g,
  p = p,
```

```
  alpha = alpha,
  test_groups = list(group1 = 1:2, group2 = 3:4),
  test_types = c(group1 = "parametric", group2 = "simes"),
  test_corr = corr_list
)
```

`graph_test_shortcut` *Perform shortcut (sequentially rejective) graphical multiple comparison procedures*

Description

Shortcut graphical multiple comparison procedures are sequentially rejective procedure based on Bretz et al. (2009). With m hypotheses, there are at most m steps to obtain all rejection decisions. These procedure are equivalent to closed graphical multiple comparison procedures using Bonferroni tests for intersection hypotheses, but shortcut procedures are faster to perform. See `vignette("shortcut-testing")` for more illustration of shortcut procedures and interpretation of their outputs.

Usage

```
graph_test_shortcut(
  graph,
  p,
  alpha = 0.025,
  verbose = FALSE,
  test_values = FALSE
)
```

Arguments

`graph`	An initial graph as returned by `graph_create()`.
`p`	A numeric vector of p-values (unadjusted, raw), whose values should be between 0 & 1. The length should match the number of hypotheses in `graph`.
`alpha`	A numeric scalar of the overall significance level, which should be between 0 & 1. The default is 0.025 for one-sided hypothesis testing problems; another common choice is 0.05 for two-sided hypothesis testing problems.
`verbose`	A logical scalar specifying whether the details of intermediate update graphs should be included in results. When `verbose = TRUE`, intermediate update graphs are provided after deleting each hypothesis, which has been rejected. The default is `verbose = FALSE`.
`test_values`	A logical scalar specifying whether adjusted significance levels should be provided for each hypothesis. When `test_values = TRUE`, it provides an equivalent way of performing graphical multiple comparison procedures by comparing each p-value with its significance level. If the p-value of a hypothesis is less than or equal to its significance level, the hypothesis is rejected. The order of rejection is based on the order of adjusted p-values from the smallest to the largest. The default is `test_values = FALSE`.

Value

An S3 object of class `graph_report` with a list of 4 elements:

- `inputs` - Input parameters, which is a list of:
  - `graph` - Initial graph, *p - (Unadjusted or raw) p-values,
  - `alpha` - Overall significance level,
  - `test_groups` - Groups of hypotheses for different types of tests, which are the list of all hypotheses for `graph_test_shortcut()`,
  - `test_types` - Different types of tests, which are "bonferroni" for `graph_test_shortcut()`.
- Output parameters `outputs`, which is a list of:
  - `adjusted_p` - Adjusted p-values,
  - `rejected` - Rejected hypotheses,
  - `graph` - Updated graph after deleting all rejected hypotheses.
- `details` - Verbose outputs with intermediate updated graphs, if `verbose = TRUE`.
- `test_values` - Adjusted significance levels, if `test_values = TRUE`.

References

Bretz, F., Maurer, W., Brannath, W., and Posch, M. (2009). A graphical approach to sequentially rejective multiple test procedures. *Statistics in Medicine*, 28(4), 586-604.

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

- `graph_test_closure()` for graphical multiple comparison procedures using the closed test,
- `graph_rejection_orderings()` for all possible rejection orderings.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

p <- c(0.018, 0.01, 0.105, 0.006)
alpha <- 0.025
graph_test_shortcut(g, p, alpha)
```

graph_test_shortcut_gsd *Perform shortcut graphical multiple comparison procedures with group sequential designs*

Description

Extends graph_test_shortcut() to group sequential designs where hypotheses can be tested at multiple analyses (interim and final). At each analysis, the significance level available for each hypothesis is determined by a spending function evaluated at the information fraction. The group sequential boundaries (critical values) are computed from the spending using the joint distribution of test statistics across analyses.

The procedure supports two modes controlled by the look_back parameter:

- look_back = FALSE (default): At each analysis, rejection decisions are based on **repeated p-values** at the current analysis only. A repeated p-value at analysis k is the minimum significance level at which the group sequential boundary at analysis k would be crossed. The graphical shortcut procedure (graph_test_shortcut()) is applied at each analysis using repeated p-values, and the graph is updated before proceeding to the next analysis.
- look_back = TRUE: Rejection decisions are based on **sequential p-values**, which consider all analyses up to the current one. A sequential p-value is the minimum of repeated p-values across all analyses conducted so far. Like the default mode, the procedure processes analyses sequentially, applying graph_test_shortcut() at each analysis using sequential p-values and updating the graph before proceeding. When a hypothesis becomes testable at a later analysis (via graph update), its first_rejected_at is set to the earliest analysis where its boundary was crossed, while decision_at records the analysis where the rejection was operationally processed.

Usage

```
graph_test_shortcut_gsd(
  graph,
  p,
  alpha = 0.025,
  info_frac,
  spending_fn,
  look_back = FALSE,
  verbose = FALSE,
  test_values = FALSE
)
```

Arguments

graph	An initial graph as returned by graph_create().
p	A numeric matrix of p-values with m rows (hypotheses) and K columns (analyses), where K is the maximum number of analyses across all hypotheses. For

hypotheses not tested at every analysis, use NA for the columns without data. Each hypothesis must have at least one non-NA value.

alpha — A numeric scalar of the overall significance level, which should be between 0 & 1. The default is 0.025 for one-sided hypothesis testing problems; another common choice is 0.05 for two-sided hypothesis testing problems.

info_frac — Information fractions at each analysis. Can be:

- A numeric vector of length K — same fractions for all hypotheses. Only allowed when p contains no NA values (i.e., all hypotheses have the same number of analyses).
- A numeric matrix with m rows (hypotheses) and K columns (analyses) — different fractions per hypothesis. When p contains NA padding, info_frac must be a matrix with NA in the same positions as p.

Non-NA values must be positive and monotonically non-decreasing per hypothesis. Values greater than 1 are allowed (e.g., when more information is collected than planned). The spending functions cap the cumulative spending at alpha for information fractions at or above 1. The last non-NA value does not need to be 1, allowing the procedure to be applied up to an interim analysis.

spending_fn — Spending function(s) for computing group sequential boundaries. Can be:

- A single function — applied to all hypotheses.
- A list of m functions — one per hypothesis.

Each function must accept two arguments: alpha (significance level) and info_frac (information fraction), and return the cumulative alpha spent. Built-in options include spending_of(), spending_pocock(), spending_hsd(), and spending_linear().

look_back — A logical scalar or vector controlling the testing strategy. Can be:

- A single logical — applied to all hypotheses.
- A logical vector of length m — one per hypothesis, allowing different strategies for different hypotheses.

For hypotheses with look_back = FALSE (the default), rejection decisions at each analysis are based on repeated p-values at that analysis only. For hypotheses with look_back = TRUE, rejection decisions are based on sequential p-values which consider all analyses up to the current one. The look_back = TRUE option can lead to additional rejections because a hypothesis may have crossed its boundary at an earlier analysis but only becomes testable (via graph update) at a later analysis.

verbose — A logical scalar specifying whether to include the boundary table in results. When verbose = TRUE, a table of nominal p-value boundaries is computed for each hypothesis at all possible weights from the graph's closure (via graph_generate_weights()). This enables manual verification of rejection decisions. The default is FALSE.

test_values — A logical scalar specifying whether to include the per-analysis rejection details in results. When test_values = TRUE, the rejection sequence, analysis at which each rejection occurred, and the nominal p-value boundaries are reported. The default is test_values = FALSE.

Value

An S3 object of class `gsd_graph_report` with a list of elements:

- `inputs` - Input parameters, including the initial graph, p-values, alpha, information fractions, and spending functions.
- `outputs` - Output parameters:
  - `repeated_p` - An m x K matrix of repeated p-values at each analysis,
  - `sequential_p` - An m x K matrix of sequential p-values (cumulative minimum of repeated p-values),
  - `adjusted_p` - Adjusted p-values from the shortcut procedure (adjusted repeated p-values when `look_back = FALSE`, adjusted sequential p-values when `look_back = TRUE`),
  - `rejected` - Logical vector of rejection decisions,
  - `decision_at` - Integer vector indicating the analysis at which each hypothesis's decision was made. For rejected hypotheses, this is the analysis where the rejection was operationally processed. For non-rejected hypotheses, this is the last analysis where the hypothesis was tested,
  - `first_rejected_at` - Integer vector indicating the earliest analysis at which each hypothesis's boundary was crossed. For non-rejected hypotheses, this is `NA`. When `look_back = TRUE`, this may be earlier than `decision_at` if a hypothesis crossed its boundary at a prior analysis but only became testable at a later analysis,
  - `last_rejected_at` - Integer vector indicating the latest analysis at which each hypothesis's boundary was crossed. For non-rejected hypotheses, this is `NA`. Comparing `first_rejected_at` and `last_rejected_at` shows whether the rejection is supported by data at multiple analyses or only at a single analysis,
  - `rejection_sequence` - Character vector giving the order in which hypotheses were rejected across all analyses,
  - `graph` - Updated graph after removing all rejected hypotheses.
- `test_values` - Per-analysis details (if `test_values = TRUE`). A list of length K (one entry per analysis). Each entry is a data frame containing the hypothesis name, current weight, observed p-value, nominal boundary, and rejection decision at that analysis. Entries are `NULL` for analyses where no hypotheses are active. When `look_back = TRUE` and a hypothesis is rejected at an earlier analysis, additional rows show the nominal p-value and boundary at each prior analysis, with a `Look_back` column indicating these rows.
- `boundary_table` - Boundary lookup table (if `verbose = TRUE`). A named list with one data frame per hypothesis. Each data frame contains the columns `Weight`, `Alpha.Allocated`, and `Boundary.k` for each analysis k, showing the nominal p-value boundary at each analysis for every possible weight from the graph's closure. This table is independent of observed p-values and can be used to manually verify rejection decisions.

References

Maurer, W., and Bretz, F. (2013). Multiple testing in group sequential trials using graphical approaches. *Statistics in Biopharmaceutical Research*, 5(4), 311-320.

Zhao, Y., Liu, Q., Sun, L. Z., and Anderson, K. M. (2025). Adjusted inference for multiple testing procedure in group-sequential designs. *Biometrical Journal*, 67(1), e70020. doi:10.1002/bimj.70020

See Also

graph_test_shortcut() for the fixed-sample (non-sequential) shortcut procedure, sequential_p() for computing sequential p-values, repeated_p() for computing repeated p-values, spending_of(), spending_pocock(), spending_hsd(), spending_linear() for spending functions.

Examples

```
# A graphical procedure with two hypotheses tested at two analyses
hypotheses <- c(0.5, 0.5)
transitions <- rbind(c(0, 1), c(1, 0))
g <- graph_create(hypotheses, transitions)

# P-values at interim (50% info) and final (100% info) analyses
p <- rbind(
  H1 = c(0.024, 0.01),
  H2 = c(0.015, 0.005)
)

graph_test_shortcut_gsd(
  graph = g,
  p = p,
  alpha = 0.025,
  info_frac = c(0.5, 1),
  spending_fn = spending_of
)

# With look_back = TRUE (sequential p-values)
graph_test_shortcut_gsd(
  graph = g,
  p = p,
  alpha = 0.025,
  info_frac = c(0.5, 1),
  spending_fn = spending_of,
  look_back = TRUE
)

# Different spending functions per hypothesis
graph_test_shortcut_gsd(
  graph = g,
  p = p,
  alpha = 0.025,
  info_frac = c(0.5, 1),
  spending_fn = list(spending_of, spending_pocock)
)

# User-defined spending functions can also be used, e.g., wrapping
# gsDesign::sfHSD(). See vignette("group-sequential-testing") for details.

# Different information fractions per hypothesis
graph_test_shortcut_gsd(
  graph = g,
  p = p,
```

```
  alpha = 0.025,
  info_frac = rbind(c(0.5, 1), c(0.6, 1)),
  spending_fn = spending_of
)

# Different numbers of analyses per hypothesis (NA padding)
# H1 at analyses 1-2, H2 at 1-3, H3 at 2-3, H4 at 1 and 3
g4 <- graph_create(
  rep(0.25, 4),
  rbind(
    c(0, 1 / 3, 1 / 3, 1 / 3),
    c(1 / 3, 0, 1 / 3, 1 / 3),
    c(1 / 3, 1 / 3, 0, 1 / 3),
    c(1 / 3, 1 / 3, 1 / 3, 0)
  )
)
p4 <- rbind(
  H1 = c(0.024, 0.01, NA),
  H2 = c(0.015, 0.005, 0.001),
  H3 = c(NA, 0.012, 0.004),
  H4 = c(0.05, NA, 0.015)
)
# info_frac must be a matrix with NA matching p
info_frac4 <- rbind(
  H1 = c(0.5, 1, NA),
  H2 = c(1 / 3, 2 / 3, 1),
  H3 = c(NA, 0.5, 1),
  H4 = c(0.4, NA, 1)
)
graph_test_shortcut_gsd(
  graph = g4,
  p = p4,
  alpha = 0.025,
  info_frac = info_frac4,
  spending_fn = spending_of
)
```

`graph_update` *Obtain an updated graph by updating an initial graphical after deleting hypotheses*

Description

After a hypothesis is deleted, an initial graph will be updated. The deleted hypothesis will have the hypothesis weight of 0 and the transition weight of 0. Remaining hypotheses will have updated hypothesis weights and transition weights according to Algorithm 1 of Bretz et al. (2009).

Usage

```
graph_update(graph, delete)
```

Arguments

graph	An initial graph as returned by graph_create().
delete	A logical or integer vector, denoting which hypotheses to delete. A logical vector results in the "unordered mode", which means that hypotheses corresponding to TRUE in delete will be deleted. The sequence of deletion will follow the sequence of TRUE's in delete. In this case, the length of the logical vector must match the number of hypotheses in graph. An integer vector results in the "ordered mode", which means that delete specifies the sequence in which hypotheses should be deleted by indicating the location of deleted hypotheses, e.g., 1st, 2nd, etc. In this case, the integer vector can have any length, but must only contain valid hypothesis numbers (greater than 0, and less than or equal to he number of hypotheses in graph).

Value

An S3 object of class updated_graph with a list of 4 elements:

- initial_graph: The initial graph object.
- updated_graph: The updated graph object with specified hypotheses deleted.
- deleted: A numeric vector indicating which hypotheses were deleted.
- intermediate_graphs: When using the ordered mode, a list of intermediate updated graphs after each hypothesis is deleted according to the sequence specified by delete.

Sequence of deletion

When there are multiple hypotheses to be deleted from a graph, there are many sequences of deletion in which an initial graph is updated to an updated graph. If the interest is in the updated graph after all hypotheses specified by delete are deleted, this updated graph is the same no matter which sequence of deletion is used. This property has been proved by Bretz et al. (2009). If the interest is in the intermediate updated graph after each hypothesis is deleted according to the sequence specified by delete, an integer vector of delete should be specified and these detailed outputs will be provided.

References

Bretz, F., Maurer, W., Brannath, W., and Posch, M. (2009). A graphical approach to sequentially rejective multiple test procedures. *Statistics in Medicine*, 28(4), 586-604.

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

- graph_create() for the initial graph.
- graph_rejection_orderings() for possible sequences of rejections for a graphical multiple comparison procedure using shortcut testing.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

# Delete the second and third hypotheses in the "unordered mode"
graph_update(g, delete = c(FALSE, TRUE, TRUE, FALSE))

# Equivalent way in the "ordered mode" to obtain the updated graph after
# deleting the second and third hypotheses
# Additional intermediate updated graphs are also provided
graph_update(g, delete = 2:3)
```

---

gs_boundaries *Compute group sequential boundaries from an alpha spending function*

Description

Given a significance level, information fractions, and a spending function, compute the group sequential boundaries at each analysis. The boundaries are computed on the Z-scale using the recursive relationship between cumulative spending and the joint distribution of test statistics. The null hypothesis is rejected at analysis k if the test statistic $Z_k \geq b_k$.

At analysis k, the Z-scale boundary b_k satisfies

$$P(Z_1 < b_1, \ldots, Z_k < b_k) = 1 - f(\alpha, t_k),$$

where $f(\alpha, t_k)$ is the cumulative spending at information fraction t_k, and $(Z_1, \ldots, Z_k)$ follows the canonical joint distribution with mean zero and correlations given by `gs_corr()`.

Usage

```
gs_boundaries(alpha, info_frac, spending_fn, maxpts = 25000, abseps = 1e-06)
```

Arguments

`alpha`	A numeric scalar of the significance level to be spent across analyses.
`info_frac`	A numeric vector of information fractions at each analysis. Must be monotonically non-decreasing with values in (0, 1].

spending_fn	A spending function that takes two arguments: alpha (significance level) and info_frac (information fraction), and returns the cumulative alpha spent. See spending_of().
maxpts	An integer scalar for the maximum number of function values for mvtnorm::GenzBretz. The default is 25000.
abseps	A numeric scalar for the absolute error tolerance for mvtnorm::GenzBretz. The default is 1e-6.

Value

A list with elements:

- bounds_z - A numeric vector of Z-scale boundaries at each analysis.
- bounds_nominal - A numeric vector of nominal p-value boundaries at each analysis, i.e., $c_k = 1 - \Phi(b_k)$.

See Also

spending_of(), spending_pocock(), spending_hsd(), spending_linear() for spending functions, sequential_p() for sequential p-values, graph_test_shortcut_gsd() for graphical MCPs with group sequential designs, gs_corr() for the correlation matrix.

---

gs_corr *Compute the correlation matrix for group sequential test statistics*

Description

In a group sequential design, test statistics $Z_1, \ldots, Z_K$ at analyses with information fractions $t_1, \ldots, t_K$ follow the canonical joint distribution with correlation

$$\text{Cor}(Z_i, Z_j) = \sqrt{t_i/t_j}, \quad i \leq j.$$

This correlation structure arises from the independent increments property of the score process. It depends only on the information fractions, not on the specific test or endpoint.

Usage

```
gs_corr(info_frac)
```

Arguments

info_frac	A numeric vector of information fractions at each analysis. Must be positive and monotonically non-decreasing.

Value

A symmetric correlation matrix of dimension $K \times K$, where K is the length of info_frac. The diagonal entries are all 1.

See Also

gs_boundaries() which uses this correlation matrix for computing group sequential boundaries.

Examples

```
# Three equally spaced analyses
gs_corr(c(1 / 3, 2 / 3, 1))

# Two analyses at 50% and 100%
gs_corr(c(0.5, 1))
```

---

plot.initial_graph *S3 plot method for class* initial_graph

Description

The plot of an initial_graph translates the hypotheses into vertices and transitions into edges to create a network plot. Vertices are labeled with hypothesis names and hypothesis weights, and edges are labeled with transition weights. See vignette("graph-examples") for more illustration of commonly used multiple comparison procedure using graphs.

Usage

```
## S3 method for class 'initial_graph'
plot(
  x,
  ...,
  v_palette = c("#6baed6", "#cccccc"),
  layout = "grid",
  nrow = NULL,
  ncol = NULL,
  edge_curves = NULL,
  precision = 4,
  eps = NULL,
  background_color = "white",
  margins = c(1, 1, 1, 1)
)
```

Arguments

x	An object of class initial_graph to plot.
...	Other arguments passed on to igraph::plot.igraph().
v_palette	A character vector of length two specifying the colors for retained and deleted hypotheses. More extensive color customization must be done with vertex.color.
layout	An igraph layout specification (See ?igraph.plotting), or "grid", which lays out hypotheses left-to-right and top-to-bottom. nrow and ncol control the grid shape.

`nrow` An integer scalar specifying the number of rows in the vertex grid. If row and column counts are not specified, vertices will be laid out as close to a square as possible.

`ncol` An integer scalar specifying the number of columns in the vertex grid. If row and column counts are not specified, vertices will be laid out as close to a square as possible.

`edge_curves` A named numeric vector specifying the curvature of specific edges. Edge pairs (Where two vertices share an edge in each possible direction) are detected automatically and get 0.25 curvature. Adjust edges by adding an entry with name `"vertex1|vertex2`, and adjust default edge pairs curvature by adding an entry with name `"pairs"` - `edge_curves = c("pairs" = 0.5, "H1|H3" = 0.25, "H3|H4" = 0.75)`.

`precision` An integer scalar indicating the number of decimal places to display.

`eps` A numeric scalar. The transition weight of `eps` will be displayed as ϵ, which indicates edges with infinitesimally small weights. See Bretz et al. (2009) for more details.

`background_color` A character scalar specifying a background color for the whole plotting area. Passed directly to `graphics::par()` (`bg`).

`margins` A length 4 numeric vector specifying the margins for the plot. Defaults to all 1, since igraph plots tend to have large margins. It is passed directly to `graphics::par()` (`mar`).

Value

An object x of class `initial_graph`, after plotting the initial graph.

Customization of graphs

There are a few values for `igraph::plot.igraph()` that get their defaults changed for graphicalMCP. These values can still be changed by passing them as arguments to `plot.initial_graph()`. Here are the new defaults:

- `vertex.color = "#6baed6"`,
- `vertex.label.color = "black"`,
- `vertex.size = 20`,
- `edge.arrow.size = 1`,
- `edge.arrow.width = 1`,
- `edge.label.color = "black"`
- `asp = 0`.

Neither `graphicalMCP` nor `igraph` does anything about overlapping edge labels. If you run into this problem, and vertices can't practically be moved enough to avoid collisions of edge labels, using edge curves can help. `igraph` puts edge labels closer to the tail of an edge when an edge is straight, and closer to the head of an edge when it's curved. By setting an edge's curve to some very small value, an effectively straight edge can be shifted to a new position.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

Xi, D., and Bretz, F. (2019). Symmetric graphs for equally weighted tests, with application to the Hochberg procedure. *Statistics in Medicine*, 38(27), 5268-5282.

See Also

`plot.updated_graph()` for the plot method for the updated graph after hypotheses being deleted from the initial graph.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 4 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
delta <- 0.5
transitions <- rbind(
  c(0, delta, 1 - delta, 0),
  c(delta, 0, 0, 1 - delta),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)
plot(g)

# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and four secondary hypotheses (H31, H32, H41, and H42)
# See Figure 6 in Xi and Bretz (2019).
hypotheses <- c(0.5, 0.5, 0, 0, 0, 0)
epsilon <- 1e-5
transitions <- rbind(
  c(0, 0.5, 0.25, 0, 0.25, 0),
  c(0.5, 0, 0, 0.25, 0, 0.25),
  c(0, 0, 0, 0, 1, 0),
  c(epsilon, 0, 0, 0, 0, 1 - epsilon),
  c(0, epsilon, 1 - epsilon, 0, 0, 0),
  c(0, 0, 0, 1, 0, 0)
)
hyp_names <- c("H1", "H2", "H31", "H32", "H41", "H42")
g <- graph_create(hypotheses, transitions, hyp_names)

plot_layout <- rbind(
  c(0.15, 0.5),
  c(0.65, 0.5),
  c(0, 0),
  c(0.5, 0),
  c(0.3, 0),
  c(0.8, 0)
```

```
)

plot(g, layout = plot_layout, eps = epsilon, edge_curves = c(pairs = .5))
```

`plot.updated_graph` *S3 plot method for the class* `updated_graph`

Description

Plotting an updated graph is a *very* light wrapper around `plot.initial_graph()`, only changing the default vertex color to use gray for deleted hypotheses.

Usage

```
## S3 method for class 'updated_graph'
plot(x, ...)
```

Arguments

`x`	An object of class `updated_graph` to plot.
`...`	Arguments passed on to `plot.initial_graph`

`v_palette` A character vector of length two specifying the colors for retained and deleted hypotheses. More extensive color customization must be done with `vertex.color`.

`layout` An igraph layout specification (See `?igraph.plotting`), or `"grid"`, which lays out hypotheses left-to-right and top-to-bottom. `nrow` and `ncol` control the grid shape.

`nrow` An integer scalar specifying the number of rows in the vertex grid. If row and column counts are not specified, vertices will be laid out as close to a square as possible.

`ncol` An integer scalar specifying the number of columns in the vertex grid. If row and column counts are not specified, vertices will be laid out as close to a square as possible.

`edge_curves` A named numeric vector specifying the curvature of specific edges. Edge pairs (Where two vertices share an edge in each possible direction) are detected automatically and get 0.25 curvature. Adjust edges by adding an entry with name `"vertex1|vertex2`, and adjust default edge pairs curvature by adding an entry with name `"pairs"` - `edge_curves = c("pairs" = 0.5, "H1|H3" = 0.25, "H3|H4" = 0.75)`.

`precision` An integer scalar indicating the number of decimal places to display.

`eps` A numeric scalar. The transition weight of `eps` will be displayed as ϵ, which indicates edges with infinitesimally small weights. See Bretz et al. (2009) for more details.

background_color A character scalar specifying a background color for the whole plotting area. Passed directly to graphics::par() (bg).

margins A length 4 numeric vector specifying the margins for the plot. Defaults to all 1, since igraph plots tend to have large margins. It is passed directly to graphics::par() (mar).

Value

An object x of class updated_graph, after plotting the updated graph.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

plot.initial_graph() for the plot method for the initial graph.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

# Delete the second and third hypotheses in the "unordered mode"
plot(
  graph_update(
    g,
    c(FALSE, TRUE, TRUE, FALSE)
  ),
  layout = "grid"
)
```

---

print.graph_report *S3 print method for the class* graph_report

---

Description

A printed graph_report displays the initial graph, p-values and significance levels, rejection decisions, and optional detailed test results.

Usage

```
## S3 method for class 'graph_report'
print(x, ..., precision = 4, indent = 2, rows = 10)
```

Arguments

x	An object of class graph_report to print.
...	Other values passed on to other methods (currently unused)
precision	An integer scalar indicating the number of decimal places to to display.
indent	An integer scalar indicating how many spaces to indent results.
rows	An integer scalar indicating how many rows of detailed test results to print.

Value

An object x of class graph_report, after printing the report of conducting a graphical multiple comparison procedure.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

p <- c(0.018, 0.01, 0.105, 0.006)
alpha <- 0.025
graph_test_shortcut(g, p, alpha)
```

`print.gsd_graph_report` *S3 print method for the class* `gsd_graph_report`

Description

A printed `gsd_graph_report` displays:

- **Test parameters**: the initial graph, alpha, information fractions, p-values, spending functions, and per-hypothesis look_back settings.
- **Test summary**: adjusted p-values, rejection decisions, the analysis at which each decision was made (`Decision.at`), the earliest analysis at which the boundary was crossed (`First.Rej.at`), look_back status, and the rejection sequence.
- **Per-analysis details** (if `test_values = TRUE`): nominal p-values, boundaries, and rejection decisions at each analysis. For hypotheses rejected via look_back, additional rows show the boundary crossing at earlier analyses, marked with * and a footnote.
- **Boundary table** (if `verbose = TRUE`): nominal p-value boundaries for all possible hypothesis weights from the graph's closure, enabling manual verification of rejection decisions.

Usage

```
## S3 method for class 'gsd_graph_report'
print(x, ..., precision = 6, indent = 2)
```

Arguments

`x`	An object of class `gsd_graph_report` to print.
`...`	Other values passed on to other methods (currently unused).
`precision`	An integer scalar indicating the number of decimal places to display.
`indent`	An integer scalar indicating how many spaces to indent results.

Value

An object x of class `gsd_graph_report`, invisibly.

References

Maurer, W., and Bretz, F. (2013). Multiple testing in group sequential trials using graphical approaches. *Statistics in Biopharmaceutical Research*, 5(4), 311-320.

Examples

```
hypotheses <- c(0.5, 0.5)
transitions <- rbind(c(0, 1), c(1, 0))
g <- graph_create(hypotheses, transitions)

p <- rbind(
  H1 = c(0.024, 0.01),
  H2 = c(0.015, 0.005)
)

graph_test_shortcut_gsd(
  graph = g,
  p = p,
  alpha = 0.025,
  info_frac = c(0.5, 1),
  spending_fn = spending_of
)
```

---

print.initial_graph *S3 print method for the class* initial_graph

---

Description

A printed initial_graph displays a header stating "Initial graph", hypothesis weights, and transition weights.

Usage

```
## S3 method for class 'initial_graph'
print(x, ..., precision = 4, indent = 0)
```

Arguments

x	An object of class initial_graph to print.
...	Other values passed on to other methods (currently unused).
precision	An integer scalar indicating the number of decimal places to to display.
indent	An integer scalar indicating how many spaces to indent results.

Value

An object x of class initial_graph, after printing the initial graph.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

print.updated_graph() for the print method for the updated graph after hypotheses being deleted from the initial graph.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
hyp_names <- c("H11", "H12", "H21", "H22")
g <- graph_create(hypotheses, transitions, hyp_names)
g
```

---

print.power_report *S3 print method for the class* power_report

---

Description

A printed power_report displays the initial graph, testing and simulation options, power outputs, and optional detailed simulations and test results.

Usage

```
## S3 method for class 'power_report'
print(x, ..., precision = 4, indent = 2, rows = 10)
```

Arguments

x	An object of the class power_report to print
...	Other values passed on to other methods (currently unused)
precision	An integer scalar indicating the number of decimal places to to display.
indent	An integer scalar indicating how many spaces to indent results.
rows	An integer scalar indicating how many rows of detailed test results to print.

Value

An object x of the class power_report, after printing the report of conducting power simulations based on a graphical multiple comparison procedure.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011a). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

Bretz, F., Maurer, W., and Hommel, G. (2011b). Test and power considerations for multiple endpoint analyses using sequentially rejective graphical procedures. *Statistics in Medicine*, 30(13), 1489-1501.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 4 in Bretz et al. (2011).
alpha <- 0.025
hypotheses <- c(0.5, 0.5, 0, 0)
delta <- 0.5
transitions <- rbind(
  c(0, delta, 1 - delta, 0),
  c(delta, 0, 0, 1 - delta),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)

marginal_power <- c(0.8, 0.8, 0.7, 0.9)
corr1 <- matrix(0.5, nrow = 2, ncol = 2)
diag(corr1) <- 1
corr <- rbind(
  cbind(corr1, 0.5 * corr1),
  cbind(0.5 * corr1, corr1)
)
success_fns <- list(
  # Probability to reject both H1 and H2
  `H1andH2` = function(x) x[1] & x[2],
  # Probability to reject both (H1 and H3) or (H2 and H4)
  `(H1andH3)or(H2andH4)` = function(x) (x[1] & x[3]) | (x[2] & x[4])
)
set.seed(1234)
# Bonferroni tests
# Reduce the number of simulations to save time for package compilation
power_output <- graph_calculate_power(
  g,
  alpha,
  sim_corr = corr,
  sim_n = 1e2,
  power_marginal = marginal_power,
  sim_success = success_fns
)
```

print.updated_graph *S3 print method for the class* updated_graph

Description

A printed updated_graph displays the initial graph, the (final) updated graph, and the sequence of intermediate updated graphs after hypotheses are deleted (if available).

Usage

```
## S3 method for class 'updated_graph'
print(x, ..., precision = 6, indent = 2)
```

Arguments

x	An object of the class updated_graph to print.
...	Other values passed on to other methods (currently unused).
precision	An integer scalar indicating the number of decimal places to to display.
indent	An integer scalar indicating how many spaces to indent results.

Value

An object x of the class updated_graph, after printing the updated graph.

References

Bretz, F., Posch, M., Glimm, E., Klinglmueller, F., Maurer, W., and Rohmeyer, K. (2011a). Graphical approaches for multiple comparison procedures using weighted Bonferroni, Simes, or parametric tests. *Biometrical Journal*, 53(6), 894-913.

See Also

print.initial_graph() for the print method for the initial graph.

Examples

```
# A graphical multiple comparison procedure with two primary hypotheses (H1
# and H2) and two secondary hypotheses (H3 and H4)
# See Figure 1 in Bretz et al. (2011).
hypotheses <- c(0.5, 0.5, 0, 0)
transitions <- rbind(
  c(0, 0, 1, 0),
  c(0, 0, 0, 1),
  c(0, 1, 0, 0),
  c(1, 0, 0, 0)
)
g <- graph_create(hypotheses, transitions)
```

```
# Delete the second and third hypotheses in the "unordered mode"
graph_update(g, delete = c(FALSE, TRUE, TRUE, FALSE))

# Equivalent way in the "ordered mode" to obtain the updated graph after
# deleting the second and third hypotheses
# Additional intermediate updated graphs are also provided
graph_update(g, delete = 2:3)
```

`repeated_p` *Calculate the repeated p-value for a single hypothesis at a given analysis*

Description

A repeated p-value at analysis k is the minimum significance level at which the group sequential boundary at analysis k would be crossed. Unlike the sequential p-value, which considers all analyses up to k, the repeated p-value only considers the boundary at analysis k itself.

The sequential p-value equals the minimum of repeated p-values across analyses: $\tilde{p}_k = \min_{l=1}^{k} \hat{p}_l$, where $\hat{p}_l$ is the repeated p-value at analysis l.

Usage

```
repeated_p(
  p,
  info_frac,
  spending_fn,
  tol = 1e-06,
  maxpts = 25000,
  abseps = 1e-06
)
```

Arguments

`p`	A numeric vector of p-values at each analysis for a single hypothesis. The length must match the length of `info_frac`. All values must be non-missing and between 0 and 1.
`info_frac`	A numeric vector of information fractions at each analysis. Values must be in (0, 1] and monotonically non-decreasing. The length should match the length of `p`.
`spending_fn`	A spending function. Must accept two arguments: `alpha` (total significance level) and `info_frac` (information fraction), and return the cumulative alpha spent. Built-in options include `spending_of()`, `spending_pocock()`, `spending_hsd()`, and `spending_linear()`.
`tol`	A numeric scalar for the tolerance of the root-finding algorithm. The default is `1e-6`.

maxpts	An integer scalar for the maximum number of function values for mvtnorm::GenzBretz. The default is 25000.
abseps	A numeric scalar for the absolute error tolerance for mvtnorm::GenzBretz. The default is 1e-6.

Details

For a hypothesis tested at analyses $k = 1, \ldots, K$ with p-values $p^{(k)}$ and information fractions $t^{(k)}$, the repeated p-value at analysis K is the minimum $\hat{p}$ such that the observed p-value $p^{(K)}$ crosses the group sequential boundary $c_K(\hat{p})$ at that analysis:

$$\hat{p}_K = \min\{\alpha : p^{(K)} \leq c_K(\alpha)\}.$$

Note that computing the boundary $c_K(\alpha)$ requires knowledge of all previous information fractions $t^{(1)}, \ldots, t^{(K)}$ because the boundary at analysis K depends on the cumulative spending and the joint distribution of test statistics.

The repeated p-value is found using stats::uniroot() on the function $g(\alpha) = z_K - b_K(\alpha)$, where $z_K = \Phi^{-1}(1 - p^{(K)})$ is the observed Z-statistic at analysis K and $b_K(\alpha)$ is the Z-scale boundary.

Value

A numeric scalar of the repeated p-value at the last analysis in the input vectors.

References

Maurer, W., and Bretz, F. (2013). Multiple testing in group sequential trials using graphical approaches. *Statistics in Biopharmaceutical Research*, 5(4), 311-320.

See Also

sequential_p() for the sequential p-value (minimum repeated p-value), gs_boundaries() for computing group sequential boundaries, graph_test_shortcut_gsd() for graphical multiple comparison procedures with group sequential designs.

Examples

```
# Repeated p-value at the second analysis (interim at 50%, final at 100%)
repeated_p(
  p = c(0.024, 0.01),
  info_frac = c(0.5, 1),
  spending_fn = spending_of
)

# Compare with sequential p-value (which is the minimum repeated p-value)
sequential_p(
  p = c(0.024, 0.01),
  info_frac = c(0.5, 1),
  spending_fn = spending_of
)
```

```
# Repeated p-values at each analysis
# Analysis 1
repeated_p(
  p = 0.05,
  info_frac = 0.3,
  spending_fn = spending_of
)

# Analysis 2
repeated_p(
  p = c(0.05, 0.02),
  info_frac = c(0.3, 0.7),
  spending_fn = spending_of
)
```

`sequential_p` *Calculate the sequential p-value for a single hypothesis*

Description

A sequential p-value is the minimum significance level at which a group sequential boundary would be crossed at any analysis up to and including the current one. It is computed using the group sequential boundaries derived from the spending function and the joint distribution of test statistics across analyses.

Sequential p-values are used in graphical multiple comparison procedures for group sequential designs. They allow the separation of the group sequential testing (handled by the spending function and boundaries) from the multiplicity adjustment (handled by the graph). See Maurer and Bretz (2013) for details.

Usage

```
sequential_p(
  p,
  info_frac,
  spending_fn,
  tol = 1e-06,
  maxpts = 25000,
  abseps = 1e-06
)
```

Arguments

`p`	A numeric vector of p-values at each analysis for a single hypothesis. The length must match the length of `info_frac`. All values must be non-missing and between 0 and 1.
`info_frac`	A numeric vector of information fractions at each analysis. Values must be in (0, 1] and monotonically non-decreasing. The length should match the length of `p`.

spending_fn	A spending function. Must accept two arguments: alpha (total significance level) and info_frac (information fraction), and return the cumulative alpha spent at information fraction t. Built-in options include spending_of(), spending_pocock(), spending_hsd(), and spending_linear().
tol	A numeric scalar for the tolerance of the root-finding algorithm. The default is 1e-6.
maxpts	An integer scalar for the maximum number of function values for mvtnorm::GenzBretz. The default is 25000.
abseps	A numeric scalar for the absolute error tolerance for mvtnorm::GenzBretz. The default is 1e-6.

Details

For a hypothesis tested at analyses $k = 1, \ldots, K$ with p-values $p^{(k)}$ and information fractions $t^{(k)}$, the sequential p-value is the minimum $\tilde{p}$ such that for some analysis k, the observed p-value $p^{(k)}$ crosses the group sequential boundary $c_k(\tilde{p})$ derived from the spending function:

$$\tilde{p} = \min\{\alpha : p^{(k)} \leq c_k(\alpha) \text{ for some } k\},$$

where $c_k(\alpha)$ is the nominal p-value boundary at analysis k when the total significance level is α.

The boundary $c_k(\alpha)$ is computed from the spending function $f(\alpha, t)$ using the joint distribution of test statistics. Specifically, the Z-scale boundary b_k satisfies

$$P(Z_1 < b_1, \ldots, Z_k < b_k) = 1 - f(\alpha, t_k),$$

and $c_k = 1 - \Phi(b_k)$. Note that $c_k \neq f(\alpha, t_k) - f(\alpha, t_{k-1})$ for $k > 1$ due to the correlation between test statistics across analyses.

The sequential p-value is found using stats::uniroot() on the function $g(\alpha) = \max_k(z_k - b_k(\alpha))$, where $z_k = \Phi^{-1}(1 - p^{(k)})$ is the observed Z-statistic and $b_k(\alpha)$ is the Z-scale boundary. This function is monotonically increasing in α since boundaries become less stringent as α increases.

Value

A numeric scalar of the sequential p-value.

References

Maurer, W., and Bretz, F. (2013). Multiple testing in group sequential trials using graphical approaches. *Statistics in Biopharmaceutical Research*, 5(4), 311-320.

Liu, Q., and Anderson, K. M. (2008). On adaptive extensions of group sequential trials for clinical investigations. *Journal of the American Statistical Association*, 103(484), 1621-1630.

See Also

gs_boundaries() for computing group sequential boundaries, graph_test_shortcut_gsd() for graphical multiple comparison procedures with group sequential designs, spending_of(), spending_pocock(), spending_hsd(), spending_linear() for spending functions.

Examples

```
# A hypothesis tested at two analyses (interim at 50% and final at 100%)
sequential_p(
  p = c(0.024, 0.01),
  info_frac = c(0.5, 1),
  spending_fn = spending_of
)

# Sequential p-value with Pocock spending
sequential_p(
  p = c(0.024, 0.01),
  info_frac = c(0.5, 1),
  spending_fn = spending_pocock
)

# Sequential p-value updates as more analyses are conducted
# After analysis 1 only
sequential_p(
  p = 0.05,
  info_frac = 0.3,
  spending_fn = spending_of
)

# After analyses 1 and 2
sequential_p(
  p = c(0.05, 0.02),
  info_frac = c(0.3, 0.7),
  spending_fn = spending_of
)

# After all three analyses
sequential_p(
  p = c(0.05, 0.02, 0.01),
  info_frac = c(0.3, 0.7, 1),
  spending_fn = spending_of
)
```

---

spending_of *Alpha spending functions for group sequential designs*

Description

Alpha spending functions determine how the total significance level (alpha) is allocated across interim and final analyses in a group sequential design. Given the total alpha and the information fraction(s) at one or more analyses, a spending function returns the cumulative alpha spent at each information fraction.

Four commonly used spending functions are provided:

- `spending_of()` for the Lan-DeMets O'Brien-Fleming approximation,

- `spending_pocock()` for the Lan-DeMets Pocock approximation,
- `spending_hsd()` for the Hwang-Shih-DeCani family,
- `spending_linear()` for linear (uniform) spending.

Usage

```
spending_of(alpha, info_frac)

spending_pocock(alpha, info_frac)

spending_hsd(alpha, info_frac, gamma = -4)

spending_linear(alpha, info_frac)
```

Arguments

`alpha`	A numeric scalar of the total significance level to be spent. Must be between 0 and 1.
`info_frac`	A numeric scalar or vector of information fractions. Values must be non-negative. When `info_frac = 0`, the spending is 0. When `info_frac >= 1`, the spending is capped at `alpha`.
`gamma`	A numeric scalar for the gamma parameter of the Hwang-Shih-DeCani spending function. Common choices are `gamma = -4` (approximates O'Brien-Fleming), `gamma = 1` (approximates Pocock), and `gamma = 0` (linear spending). The default is `gamma = -4`.

Details

All spending functions satisfy the following properties:

- $f(\alpha, 0) = 0$,
- $f(\alpha, 1) = \alpha$,
- $f(\alpha, t)$ is non-decreasing in t.

The cumulative alpha spent at analysis k is $f(\alpha, t_k)$, and the incremental spending is

$$\Delta\alpha_k = f(\alpha, t_k) - f(\alpha, t_{k-1}).$$

Note that the incremental spending is *not* the nominal significance level (boundary) at analysis k. The boundary must be derived from the spending using the joint distribution of test statistics across analyses. See `sequential_p()` and `graph_test_shortcut_gsd()` for details.

Value

A numeric vector the same length as `info_frac` of cumulative alpha spent at each information fraction.

Spending function formulas

- **O'Brien-Fleming** (`spending_of`):

$$f(\alpha, t) = 2\left(1 - \Phi\left(\frac{\Phi^{-1}(1-\alpha/2)}{\sqrt{t}}\right)\right).$$

  This is the Lan-DeMets approximation to O'Brien-Fleming boundaries. It is very conservative at early analyses and spends most of the alpha at the final analysis.

- **Pocock** (`spending_pocock`):

$$f(\alpha, t) = \alpha \cdot \ln(1 + (e-1) \cdot t).$$

  This spends alpha more evenly across analyses compared to O'Brien-Fleming.

- **Hwang-Shih-DeCani** (`spending_hsd`):

$$f(\alpha, t) = \alpha \cdot \frac{1 - e^{-\gamma t}}{1 - e^{-\gamma}}, \quad \gamma \neq 0,$$

$$f(\alpha, t) = \alpha \cdot t, \quad \gamma = 0.$$

  With `gamma = -4`, it approximates O'Brien-Fleming; with `gamma = 1`, it approximates Pocock.

- **Linear** (`spending_linear`):

$$f(\alpha, t) = \alpha \cdot t.$$

References

Lan, K. K. G., and DeMets, D. L. (1983). Discrete sequential boundaries for clinical trials. *Biometrika*, 70(3), 659-663.

Hwang, I. K., Shih, W. J., and De Cani, J. S. (1990). Group sequential designs using a family of type I error probability spending functions. *Statistics in Medicine*, 9(12), 1439-1445.

Examples

```
# O'Brien-Fleming spending at 50% information
spending_of(0.025, 0.5)

# Cumulative spending across analyses (vectorized)
spending_of(0.025, c(0, 0.5, 1))

# Compare spending functions at information fractions (1/3, 2/3, 1)
spending_of(0.025, c(1 / 3, 2 / 3, 1))
spending_pocock(0.025, c(1 / 3, 2 / 3, 1))
spending_hsd(0.025, c(1 / 3, 2 / 3, 1), gamma = -4)
spending_linear(0.025, c(1 / 3, 2 / 3, 1))

# User-defined spending function: piecewise combination.
# Use O'Brien-Fleming for the first half of alpha (conservative at
# early analyses), and Pocock for the second half (more aggressive).
# This can be useful when a hypothesis starts with a small weight
# (OBF spending) and later receives additional weight via graph
# propagation (Pocock spending for the increment).
```

```
spending_piecewise <- function(alpha, info_frac, threshold = 0.0125) {
  spending_of(pmin(alpha, threshold), info_frac) +
    spending_pocock(pmax(alpha - threshold, 0), info_frac)
}
spending_piecewise(0.025, c(1 / 3, 2 / 3, 1))
# Compare: alpha = 0.0125 uses only OBF
spending_piecewise(0.0125, c(1 / 3, 2 / 3, 1))
spending_of(0.0125, c(1 / 3, 2 / 3, 1))
# Pocock spending at 50% information
spending_pocock(0.025, 0.5)
# Hwang-Shih-DeCani spending at 50% information
spending_hsd(0.025, 0.5, gamma = -4)
spending_hsd(0.025, 0.5, gamma = 1)
spending_hsd(0.025, 0.5, gamma = 0)
# Linear spending at 50% information
spending_linear(0.025, 0.5)
```

`spending_with_time` *Create a spending function with a custom spending time*

Description

Wraps an existing spending function to use a fixed **spending time** instead of the information fractions passed to it at runtime. This controls only the alpha allocation schedule. The correlation structure of the test statistics is determined separately by the `info_frac` argument in `graph_test_shortcut_gsd()` (via `gs_corr()`), not by the spending function.

This is useful in two common scenarios:

- **Subgroup analyses**: all-subjects hypotheses use subgroup event fractions as spending time (controlling how alpha is allocated across analyses), while `info_frac` in `graph_test_shortcut_gsd()` uses all-subjects event fractions (controlling the correlation structure).
- **Monitoring with changed final information**: when the actual total information at the final analysis differs from the planned total, the planned information fractions are used as spending time to preserve the alpha allocation at earlier analyses, while `info_frac` in `graph_test_shortcut_gsd()` uses the actual information fractions for the correlation structure.

Usage

```
spending_with_time(spending_fn, spending_time, info_frac = NULL)
```

Arguments

`spending_fn`	A spending function to wrap. Must accept two arguments: `alpha` (significance level) and `info_frac` (information fraction), and return the cumulative alpha spent.
`spending_time`	A numeric vector of spending time values. These replace the `info_frac` argument when the wrapped function is called. May contain `NA` for analyses that are skipped (e.g., a hypothesis not tested at a particular analysis). The last non-`NA` value should be 1 if the final analysis has been specified.

`info_frac` An optional numeric vector of information fractions with the same length as `spending_time`. If provided, the `NA` positions are validated to match those in `spending_time`. This ensures that the spending time and information fraction structures are consistent.

Value

A function with the same signature as `spending_fn` — `function(alpha, info_frac)` — that internally uses `spending_time` instead of `info_frac` for alpha allocation.

See Also

`spending_of()`, `spending_pocock()`, `spending_hsd()`, `spending_linear()` for built-in spending functions, `graph_test_shortcut_gsd()` for the graphical procedure with group sequential designs.

Examples

```
# --- Subgroup spending time ---
# Without spending_with_time, spending_of() uses info_frac for spending:
info_frac_all <- c(529 / 800, 700 / 800, 1) # all-subjects fractions
spending_of(0.01, info_frac_all)

# With spending_with_time, spending uses subgroup fractions instead.
# The info_frac passed at runtime is ignored by the spending function;
# it is only used by gs_boundaries()/graph_test_shortcut_gsd() for
# the correlation structure.
spending_time_sub <- c(185 / 295, 245 / 295, 1) # subgroup fractions
spending_with_time(spending_of, spending_time_sub)

# --- Monitoring with changed final information ---
# Planned: 295 OS events at 3 analyses (185, 245, 295 events).
# spending_time uses planned fractions for interim analyses and 1
# for the final analysis.
spending_monitor <- spending_with_time(
  spending_of,
  spending_time = c(185 / 295, 245 / 295, 1)
)

# Overrunning (310 events) or underrunning (280 events):
# spending_time is the same in both cases — it uses planned fractions
# for interim analyses and 1 for the final analysis, because alpha
# spent has been fixed for interim analyses. The actual info_frac
# (which differs between overrunning and underrunning) only affects
# the correlation structure in gs_boundaries()/graph_test_shortcut_gsd().
spending_monitor(0.01, c(185 / 295, 245 / 295, 1))

# --- Skipped analyses (NA in spending_time) ---
# If a hypothesis is not tested at analysis 2, both spending_time and
# info_frac have NA at that position. The output also has NA there.
spending_skip <- spending_with_time(
  spending_of,
```

```
  spending_time = c(185 / 295, NA, 1),
  info_frac = c(185 / 295, NA, 1)
)
spending_skip(0.01, c(185 / 295, NA, 1))
```

`spending_wt` *Wang-Tsiatis spending function*

Description

Computes the implied cumulative alpha spending from the Wang-Tsiatis family of group sequential boundaries. The Wang-Tsiatis boundaries at analysis k with information fraction t_k are defined as:

$$c_k = C \cdot t_k^{\Delta - 0.5},$$

where Δ is the shape parameter and C is a constant calibrated so that the overall Type I error equals α.

Special cases:

- $\Delta = 0.5$: Pocock boundaries (equal Z-scale boundaries across analyses).
- $\Delta = 0$: O'Brien-Fleming boundaries (very conservative at early analyses).
- $0 < \Delta < 0.5$: intermediate between O'Brien-Fleming and Pocock.

Unlike the Lan-DeMets approximations (`spending_of()`, `spending_pocock()`), this function computes the **exact** boundaries from the Wang-Tsiatis family and derives the implied spending. It is computationally more expensive because it requires root-finding and multivariate normal integration at each call.

Usage

```
spending_wt(alpha, info_frac, delta = 0.5, maxpts = 25000, abseps = 1e-06)
```

Arguments

`alpha`	A numeric scalar of the total significance level.
`info_frac`	A numeric vector of information fractions at each analysis. Must be non-negative, with at most one value ≥ 1. The last value must be ≥ 1 (i.e., the final analysis must be included), because the Wang-Tsiatis constant C is calibrated over the full set of analyses.
`delta`	A numeric scalar for the shape parameter Δ. The default is `0.5` (Pocock). Use `0` for O'Brien-Fleming.
`maxpts`	An integer scalar for the maximum number of function values for `mvtnorm::GenzBretz()`. The default is 25000.
`abseps`	A numeric scalar for the absolute error tolerance for `mvtnorm::GenzBretz()`. The default is 1e-6.

Value

A numeric vector the same length as `info_frac` of cumulative alpha spent at each information fraction.

References

Wang, S. K., and Tsiatis, A. A. (1987). Approximately optimal one-parameter boundaries for group sequential trials. *Biometrics*, 43(1), 193-199.

See Also

`spending_of()` and `spending_pocock()` for the Lan-DeMets approximations, `gs_boundaries()` for computing boundaries from spending functions, `graph_test_shortcut_gsd()` for the graphical procedure.

Examples

```
# Exact O'Brien-Fleming (delta = 0)
spending_wt(0.025, c(0.5, 1), delta = 0)

# Exact Pocock (delta = 0.5)
spending_wt(0.025, c(0.5, 1), delta = 0.5)

# Intermediate (delta = 0.25)
spending_wt(0.025, c(1 / 3, 2 / 3, 1), delta = 0.25)

# Compare with Lan-DeMets approximations
spending_of(0.025, c(1 / 3, 2 / 3, 1)) # Lan-DeMets OBF approximation
spending_wt(0.025, c(1 / 3, 2 / 3, 1), 0) # Exact OBF

# Use in graph_test_shortcut_gsd (wrap to fix delta)

g <- graph_create(c(0.5, 0.5), rbind(c(0, 1), c(1, 0)))
p <- rbind(H1 = c(0.024, 0.01), H2 = c(0.015, 0.005))
graph_test_shortcut_gsd(
  graph = g, p = p, alpha = 0.025,
  info_frac = c(0.5, 1),
  spending_fn = function(a, t) spending_wt(a, t, delta = 0.25)
)
```

Index