[bonferroni\_sequential](https://urban-sniffle-p11zlpj.pages.github.io/reference/testing.html)(ss\_graph, p = pvals, alpha = .05, critical = TRUE)

#>

#> Test parameters ----------------------------------------------------------------

#> An initial graph

#>

#> --- Hypothesis weights ---

#> A1: 0.5000

#> B1: 0.5000

#> A2: 0.0000

#> B2: 0.0000

#>

#> --- Transition weights ---

#> A1 B1 A2 B2

#> A1 0.0000 0.5000 0.5000 0.0000

#> B1 0.5000 0.0000 0.0000 0.5000

#> A2 0.0000 1.0000 0.0000 0.0000

#> B2 1.0000 0.0000 0.0000 0.0000

#>

#> Global alpha = 0.05

#>

#> A1 B1 A2 B2

#> Unadjusted p-values: 0.024 0.01 0.026 0.027

#>

#> Test types

#> bonferroni: (A1-B1-A2-B2)

#>

#> Global test summary ------------------------------------------------------------

#> Hypothesis Adj. P-value Reject

#> A1 0.032 TRUE

#> B1 0.020 TRUE

#> A2 0.052 FALSE

#> B2 0.052 FALSE

#>

#> Test details - Critical values -------------------------------------------------

#> step hypothesis test p <= w \* alpha res

#> 1 B1 bonferroni 0.010 <= 0.50 \* 0.05 TRUE

#> 2 A1 bonferroni 0.024 <= 0.75 \* 0.05 TRUE

#> 3 A2 bonferroni 0.026 <= 0.50 \* 0.05 FALSE

#> 4 B2 bonferroni 0.027 <= 1.00 \* 0.05 TRUE

bonferroni\_sequential <- function(graph,

p,

alpha = .05,

critical = FALSE) {

test\_input\_val(

graph,

p,

alpha,

groups = list(seq\_along(graph$hypotheses)),

test\_types = "bonferroni",

corr = NULL,

verbose = FALSE,

critical = critical

)

initial\_graph <- graph

adj\_p\_max <- 0

adj\_p <- vector("numeric", length(graph$hypotheses))

rejected <- vector("logical", length(graph$hypotheses))

critical\_vals <- if (critical) vector("list", length(graph$hypotheses))

keep <- rep(TRUE, length(graph$hypotheses))

for (i in seq\_along(graph$hypotheses)) {

adj\_p\_subgraph <- p / graph$hypotheses

min\_index <- which.min(adj\_p\_subgraph)

adj\_p\_max <- max(adj\_p\_max, adj\_p\_subgraph[[min\_index]])

adj\_p[[min\_index]] <- min(adj\_p\_max, 1) # cap adjusted p-values at 1

rejected[[min\_index]] <- adj\_p\_max <= alpha

if (critical) {

critical\_step <- bonferroni\_test\_vals(

p[min\_index],

graph$hypotheses[min\_index],

alpha

)

critical\_step[[1]] <- i

names(critical\_step)[[1]] <- "step"

critical\_step[6:7] <- NULL

critical\_vals[[i]] <- critical\_step

}

keep[[min\_index]] <- FALSE

graph <- update\_graph(graph, keep)$updated\_graph

}

names(adj\_p) <- names(graph$hypotheses)

names(rejected) <- names(graph$hypotheses)

critical\_vals <- if (critical) list(results = do.call(rbind, critical\_vals))

structure(

list(

inputs = list(

graph = initial\_graph,

p = p,

alpha = alpha,

groups = list(seq\_along(initial\_graph$hypotheses)),

test\_types = "bonferroni",

corr = NULL

),

outputs = list(p\_adj = adj\_p, rejected = rejected),

details = NULL,

critical = critical\_vals

),

class = "graph\_report"

)

}

test\_input\_val <- function(graph,

p,

alpha,

groups = list(seq\_along(graph$hypotheses)),

test\_types = c("bonferroni"),

corr,

verbose,

critical) {

test\_opts <- c(

bonferroni = "bonferroni",

parametric = "parametric",

simes = "simes",

b = "bonferroni",

p = "parametric",

s = "simes"

)

stopifnot(

"Please test an `initial\_graph` object" = class(graph) == "initial\_graph",

"P-values must be numeric" = is.numeric(p),

"P-values must be between 0 & 1" = all(p >= 0 & p <= 1),

"Alpha must be numeric" = is.numeric(alpha),

"Please choose a single alpha level for testing" = length(alpha) == 1,

"Alpha must be between 0 & 1" = alpha >= 0 && alpha <= 1,

"Only Bonferroni, parametric, and Simes tests are currently supported" =

all(test\_types %in% test\_opts),

"Groups specification must be a list" = is.list(groups),

"Please include each hypothesis in exactly one group" =

setequal(seq\_along(graph$hypotheses), unlist(groups)) &&

length(graph$hypotheses) == length(unlist(groups)),

"Please choose one test, or one test per group" =

length(test\_types) == length(groups),

"Length of p-values & groups must match size of graph" =

unique(length(p), length(unlist(groups))) == length(graph$hypotheses),

"Verbose flag must be a length one logical" =

is.logical(verbose) && length(verbose) == 1,

"Critical flag must be a length one logical" =

is.logical(critical) && length(critical) == 1

)

missing\_corr <- any(

vapply(

seq\_along(test\_types),

function(i) {

if (test\_types[[i]] == "parametric") {

group <- groups[[i]]

missing <- any(is.na(corr[group, group])) || is.null(corr)

pos\_def <- all(eigen(corr[group, group])$values > 0)

return(missing)

} else {

return(FALSE)

}

},

logical(1)

)

)

positive\_definite\_corr <- ifelse(

!missing\_corr,

all(

vapply(

seq\_along(test\_types),

function(i) {

if (test\_types[[i]] == "parametric") {

group <- groups[[i]]

pos\_def <- all(eigen(corr[group, group])$values >= 0)

return(pos\_def)

} else {

return(TRUE)

}

},

logical(1)

)

),

TRUE

)

stopifnot(

"Correlation sub-matrix for each parametric test group must be complete" =

!missing\_corr,

"Correlation matrix must be symmetric" =

isSymmetric.matrix(corr) || is.null(corr),

"Dimensions of correlation matrix must match size of graph" =

unique(nrow(corr), ncol(corr)) == length(graph$hypotheses) ||

is.null(corr),

"Correlation matrix must be between 0 & 1" =

all((corr >= 0 & corr <= 1) | is.na(corr)) || is.null(corr),

"Correlation matrix must be positive definite" = positive\_definite\_corr

)

invisible(graph)

}

power\_input\_val <- function(graph, n, theta, corr, success) {

graph\_size <- length(graph$hypotheses)

stopifnot(

"Number of simulations must be a length one integer" =

is.numeric(n) && as.integer(n) == n && length(n) == 1,

"Mean and covariance parameters must be numeric" =

is.numeric(theta) && is.numeric(corr),

"Lengths of `marginal\_power` and `sim\_corr` must match graph size" =

unique(length(theta), nrow(corr), ncol(corr)) == graph\_size,

"Covariance matrix for simulating p-values cannot have missing values" =

!any(is.na(corr)),

"Covariance matrix for simulating p-values must be symmetric" =

isSymmetric.matrix(corr),

"Covariance matrix for simulating p-values must have diagonal all 1" =

all(diag(corr) == 1),

"'Success' hypotheses must be positive integers" =

all(as.integer(success) == success) && all(success > 0),

"'Success' hypotheses must be less than graph size, and be unique" =

length(unique(success)) == length(success) && max(success) <= graph\_size

)

invisible(graph)

}