Package ‘hierbase’

November 8, 2021

Type Package

Title Enabling Hierarchical Multiple Testing

Version 0.1.2

Description Implementation of hierarchical inference based on Meinshausen (2008).
Hierarchical testing of variable importance. Biometrika, 95(2), 265-278 and
Renaux, Buzdugan, Kalisch, and Bühlmann, (2020). Hierarchical inference for
genome-wide association studies: a view on methodology with software.
Computational Statistics, 35(1), 1-40.
The R-package 'hierbase' offers tools to perform hierarchical inference
for one or multiple data sets based on ready-to-use (group) test functions
or alternatively a user specified (group) test function.
The procedure is based on a hierarchical multiple testing
correction and controls the family-wise error rate (FWER).
The functions can easily be run in parallel.
Hierarchical inference can be applied to (low- or) high-dimensional
data sets to find significant groups or single variables (depending on
the signal strength and correlation structure) in a data-driven and
automated procedure. Possible applications can for example be found
in statistical genetics and statistical genomics.

License GPL-3

Encoding UTF-8

Depends R (>= 4.0.0)

Imports glmnet, hdi, methods, parallel, stats, SIHR

Suggests knitr, MASS, testthat

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation no

Author Claude Renaux [aut, cre],
Peter Bühlmann [ths]

Maintainer Claude Renaux `<renaux@stat.math.ethz.ch>`

Repository CRAN

Date/Publication 2021-11-08 12:00:02 UTC

R topics documented:

advance_hierarchy 2
cluster_positions 5
cluster_vars 7
hierbase 10
print.hierBase 10
run_hierarchy 12

Index **17**

`advance_hierarchy` *Hierarchical Testing*

Description

Hierarchical testing for a given build-in test function.

Usage

```
advance_hierarchy(
  x,
  y,
  dendr,
  test = c("QF", "hdi", "hdi.logistic", "F", "logistic"),
  clvar = NULL,
  alpha = 0.05,
  global.test = TRUE,
  hier.adj = FALSE,
  mt.adj = c("dpBF", "none"),
  agg.method = c("Tippett", "Stouffer"),
  verbose = FALSE,
  sort.parallel = TRUE,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1L,
  cl = NULL
)
```

Arguments

x	a matrix or list of matrices for multiple data sets. The matrix or matrices have to be of type numeric and are required to have column names / variable names. The rows and the columns represent the observations and the variables, respectively.
y	a vector, a matrix with one column, or list of the aforementioned objects for multiple data sets. The vector, vectors, matrix, or matrices have to be of type numeric.
dendr	the output of one of the functions `cluster_vars` or `cluster_positions`.

`test`	a character string naming a 'build-in' group test function. See the 'Details' section.
`clvar`	a matrix or list of matrices of control variables.
`alpha`	the significant level at which the FWER is controlled.
`global.test`	a logical value indicating whether the global test should be performed.
`hier.adj`	a logical value indicating whether the p-values can only increase when going down some given branch. Strong FWER control holds as well if the argument is set to FALSE which is the default option.
`mt.adj`	type of multiple testing correction to be used; either `"dpBF"` (depth-wise Bonferroni multiple adjustment) or `"none"` (no adjustment). See the 'Details' section.
`agg.method`	a character string naming an aggregation method which aggregates the p-values over the different data sets for a given cluster; either `"Tippett"` (Tippett's rule) or `"Stouffer"` (Stouffer's rule). This argument is only relevant if multiple data sets are specified in the function call.
`verbose`	a logical value indicating whether the progress of the computation should be printed in the console.
`sort.parallel`	a logical indicating whether the blocks should be sorted with respect to the size of the block. This can reduce the run time for parallel computation.
`parallel`	type of parallel computation to be used. See the 'Details' section.
`ncpus`	number of processes to be run in parallel.
`cl`	an optional **parallel** or **snow** cluster used if `parallel = "snow"`. If not supplied, a cluster on the local machine is created.

Details

Hierarchical testing is performed by going top down through the hierarchical tree. Testing in some branch only continues if at least one child of a given cluster is significant. The function `advance_hierarchy` requires the output of one of the functions `cluster_vars` or `cluster_positions` as an input (argument `dendr`).

The user can choose one of the 'build-in' group test functions that is applied to every group which is tested. The default is `test = "QF"` (inference for quadratic functionals in linear regression; see function `QF` in the R package **SIHR**). Alternatively, there are `"hdi"` (de-biased Lasso for linear regression; see function `lasso.proj` in the R package **hdi**), `"hdi.logistic"` (de-biased Lasso for logistic regression; see function `lasso.proj` in the R package **hdi**), `"F"` (classical partial F-Test for low-dimensional data; see function `anova`), and `"logistic"` (likelihood ratio test for logistic regression for low-dimensional data; see function `anova`).

If one of the 'build-in' group test functions `"QF"`, `"hdi"`, or `"hdi.logistic"` is applied and control variables are specified using the argument `clvar`, then those variables are included in the model, there is an L1-penalty in the Lasso imposed on them, and obviously those covariates are not tested in the hierarchical procedure.

The user can specify which hierarchical multiple testing adjustment for the hierarchical procedure is applied. The default is `"dpBF"` (depth-wise Bonferroni multiple adjustment). Alternatively, the user can choose `"none"` (no adjustment). The hierarchical multiple testing adjustment `"dpBF"` guarantees strong family-wise error control if the group test, which is applied for testing a given group, controls the type I error.

If the argument `block` was supplied for the building of the hierarchical tree (i.e. in the function call of either `cluster_vars` or `cluster_positions`), i.e. the second level of the hierarchical tree was given, the hierarchical testing step can be run in parallel across the different blocks by specifying the arguments `parallel` and `ncpus`. There is an optional argument `cl` if `parallel = "snow"`. There are three possibilities to set the argument `parallel`: `parallel = "no"` for serial evaluation (default), `parallel = "multicore"` for parallel evaluation using forking, and `parallel = "snow"` for parallel evaluation using a parallel socket cluster. It is recommended to select `RNGkind("L'Ecuyer-CMRG")` and set a seed to ensure that the parallel computing of the package `hierbase` is reproducible. This way each processor gets a different substream of the pseudo random number generator stream which makes the results reproducible if the arguments (as `sort.parallel` and `ncpus`) remain unchanged. See the vignette or the reference for more details.

Note that if Tippett's aggregation method is applied for multiple data sets, then very small p-values are set to machine precision. This is due to rounding in floating point arithmetic.

Value

The returned value is an object of class `"hierBase"`, consisting a data.frame with the result of the hierarchical testing.

The data.frame has the following columns:

`block`	`NA` or the name of the block if the significant cluster is a subcluster of the block or is the block itself.
`p.value`	The p-value of the significant cluster.
`significant.cluster`	The column names of the members of the significant cluster.

There is a `print` method for this class; see `print.hierBase`.

References

Meinshausen, N. (2008). Hierarchical testing of variable importance. Biometrika, 95(2), 265-278. Renaux, C., Buzdugan, L., Kalisch, M., and Bühlmann, P. (2020). Hierarchical inference for genome-wide association studies: a view on methodology with software. Computational Statistics, 35(1), 1-40.

See Also

`cluster_vars`, `cluster_positions`, and `run_hierarchy`.

Examples

```
## Low-dimensonal example
n <- 100
p <- 50
library(MASS)
set.seed(3)
x <- mvrnorm(n, mu = rep(0, p), Sigma = diag(p))
colnames(x) <- paste0("Var", 1:p)
beta <- rep(0, p)
beta[c(5, 20, 46)] <- 1
```

```
y <- x %*% beta + rnorm(n)

dendr1 <- cluster_vars(x = x)
set.seed(76)
sign.clusters1 <- advance_hierarchy(x = x, y = y, dendr = dendr1,
                                    test = "F")

## High-dimensional example
if (FALSE) {
  n <- 50
  p <- 80
  library(MASS)
  set.seed(3)
  x <- mvrnorm(n, mu = rep(0, p), Sigma = diag(p))
  colnames(x) <- paste0("Var", 1:p)
  beta <- rep(0, p)
  beta[c(5, 20, 46)] <- 1
  y <- x %*% beta + rnorm(n)

  dendr1 <- cluster_vars(x = x)
  set.seed(76)
  sign.clusters1 <- advance_hierarchy(x = x, y = y, dendr = dendr1,
                                    test = "QF")

  ## With block
  # I.e. second level of the hierarchical tree is specified by
  # the user. This would allow to run the code in parallel; see the 'Details'
  # section.
  # The column names of the data frame block are optional.

  block <- data.frame("var.name" = paste0("Var", 1:p),
                      "block" = rep(c(1, 2), each = p/2))
  dendr2 <- cluster_vars(x = x, block = block)
  set.seed(76)
  sign.clusters2 <- advance_hierarchy(x = x, y = y, dendr = dendr2,
                                      test = "QF")

  # Access part of the return object or result
  sign.clusters2[, "block"]
  sign.clusters2[, "p.value"]
  # Column names or variable names of the significant cluster in the first row.
  sign.clusters2[[1, "significant.cluster"]]
}
```

`cluster_positions` *Build Hierarchical Tree based on Position*

Description

Build a hierarchical tree based on the position of the variables.

Usage

```
cluster_positions(
  position,
  block = NULL,
  sort.parallel = TRUE,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1L,
  cl = NULL
)
```

Arguments

`position`	a data frame with two columns specifying the variable names and the corresponding position or a list of data frames for multiple data sets. The first column is required to contain the variable names and to be of type character. The second column is required to contain the position and to be of type numeric.
`block`	a data frame or matrix specifying the second level of the hierarchical tree. The first column is required to contain the variable names and to be of type character. The second column is required to contain the group assignment and to be a vector of type character or numeric. If not supplied, the second level is built based on the data.
`sort.parallel`	a logical indicating whether the blocks should be sorted with respect to the size of the block. This can reduce the run time for parallel computation.
`parallel`	type of parallel computation to be used. See the 'Details' section.
`ncpus`	number of processes to be run in parallel.
`cl`	an optional **parallel** or **snow** cluster used if `parallel = "snow"`. If not supplied, a cluster on the local machine is created.

Details

The hierarchical tree is built based on recursive binary partitioning of consecutive variables w.r.t. their position. The partitioning consists of splitting a given node / cluster into two children of about equal size based on the positions of the variables. If a node contains an odd number of variables, then the variable in the middle w.r.t. position is assigned to the cluster containing the closest neighbouring variable. Hence, clusters at a given depth of the binary hierarchical tree contain about the same number of variables.

If the argument `block` is supplied, i.e. the second level of the hierarchical tree is given, the function can be run in parallel across the different blocks by specifying the arguments `parallel` and `ncpus`. There is an optional argument `cl` if `parallel = "snow"`. There are three possibilities to set the argument `parallel`: `parallel = "no"` for serial evaluation (default), `parallel = "multicore"` for parallel evaluation using forking, and `parallel = "snow"` for parallel evaluation using a parallel socket cluster. It is recommended to select `RNGkind("L'Ecuyer-CMRG")` and set a seed to ensure that the parallel computing of the package `hierinf` is reproducible. This way each processor gets a different substream of the pseudo random number generator stream which makes the results reproducible if the arguments (as `sort.parallel` and `ncpus`) remain unchanged. See the vignette or the reference for more details.

Value

The returned value is an object of class "hierD", consisting of two elements, the argument "block" and the hierarchical tree "res.tree".

The element "block" defines the second level of the hierarchical tree if supplied.

The element "res.tree" contains a dendrogram for each of the blocks defined in the argument block. If the argument block is NULL (i.e. not supplied), the element contains only one dendrogram.

References

Meinshausen, N. (2008). Hierarchical testing of variable importance. Biometrika, 95(2), 265-278. Renaux, C., Buzdugan, L., Kalisch, M., and Bühlmann, P. (2020). Hierarchical inference for genome-wide association studies: a view on methodology with software. Computational Statistics, 35(1), 1-40.

See Also

cluster_vars, advance_hierarchy, and run_hierarchy.

Examples

```
# The column names of the data frames position and block are optional.
position <- data.frame("var.name" = paste0("Var", 1:500),
                       "position" = seq(from = 1, to = 1000, by = 2))
dendr1 <- cluster_positions(position = position)

block <- data.frame("var.name" = paste0("Var", 1:500),
                    "block" = rep(c(1, 2), each = 250),
                    stringsAsFactors = FALSE)
dendr2 <- cluster_positions(position = position, block = block)
```

cluster_vars *Build a Hierarchical Tree based on Hierarchical Clustering*

Description

Build a hierarchical tree based on hierarchical clustering of the variables.

Usage

```
cluster_vars(
  x = NULL,
  d = NULL,
  block = NULL,
  method = "average",
  use = "pairwise.complete.obs",
  sort.parallel = TRUE,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1L,
  cl = NULL
)
```

Arguments

`x`	a matrix or list of matrices for multiple data sets. The matrix or matrices have to be of type numeric and are required to have column names / variable names. The rows and the columns represent the observations and the variables, respectively. Either the argument `x` or `d` has to be specified.
`d`	a dissimilarity matrix. This can be either a symmetric matrix of type numeric with column and row names or an object of class `dist` with labels. Either the argument `x` or `d` has to be specified.
`block`	a data frame or matrix specifying the second level of the hierarchical tree. The first column is required to contain the variable names and to be of type character. The second column is required to contain the group assignment and to be a vector of type character or numeric. If not supplied, the second level is built based on the data.
`method`	the agglomeration method to be used for the hierarchical clustering. See `hclust` for details.
`use`	the method to be used for computing covariances in the presence of missing values. This is important for multiple data sets which do not measure exactly the same variables. If data is specified using the argument `x`, the dissimilarity matrix for the hierarchical clustering is calculated using correlation. See the 'Details' section and `cor` for all the options.
`sort.parallel`	a logical indicating whether the blocks should be sorted with respect to the size of the block. This can reduce the run time for parallel computation.
`parallel`	type of parallel computation to be used. See the 'Details' section.
`ncpus`	number of processes to be run in parallel.
`cl`	an optional **parallel** or **snow** cluster used if `parallel = "snow"`. If not supplied, a cluster on the local machine is created.

Details

The hierarchical tree is built by hierarchical clustering of the variables. Either the data (using the argument `x`) or a dissimilarity matrix (using the argument `d`) can be specified.

If one or multiple data sets are defined using the argument `x`, the dissimilarity matrix is calculated by one minus squared empirical correlation. In the case of multiple data sets, a single hierarchical tree is jointly estimated using hierarchical clustering. The argument `use` is important because missing values are introduced if the data sets do not measure exactly the same variables. The argument `use` determines how the empirical correlation is calculated.

Alternatively, it is possible to specify a user-defined dissimilarity matrix using the argument `d`.

If the argument `x` and `block` are supplied, i.e. the `block` defines the second level of the hierarchical tree, the function can be run in parallel across the different blocks by specifying the arguments parallel and ncpus. There is an optional argument cl if parallel = "snow". There are three possibilities to set the argument parallel: parallel = "no" for serial evaluation (default), parallel = "multicore" for parallel evaluation using forking, and parallel = "snow" for parallel evaluation using a parallel socket cluster. It is recommended to select RNGkind("L'Ecuyer-CMRG") and set a seed to ensure that the parallel computing of the package hierinf is reproducible. This way each processor gets a different substream of the pseudo random number generator stream which makes the results reproducible if the arguments (as sort.parallel and ncpus) remain unchanged. See the vignette or the reference for more details.

Value

The returned value is an object of class "hierD", consisting of two elements, the argument "block" and the hierarchical tree "res.tree".

The element "block" defines the second level of the hierarchical tree if supplied.

The element "res.tree" contains a dendrogram for each of the blocks defined in the argument block. If the argument block is NULL (i.e. not supplied), the element contains only one dendrogram.

References

Meinshausen, N. (2008). Hierarchical testing of variable importance. Biometrika, 95(2), 265-278. Renaux, C., Buzdugan, L., Kalisch, M., and Bühlmann, P. (2020). Hierarchical inference for genome-wide association studies: a view on methodology with software. Computational Statistics, 35(1), 1-40.

See Also

cluster_positions, advance_hierarchy, and run_hierarchy.

Examples

```
library(MASS)
x <- mvrnorm(50, mu = rep(0, 100), Sigma = diag(100))
colnames(x) <- paste0("Var", 1:100)
dendr1 <- cluster_vars(x = x)

# The column names of the data frame block are optional.
block <- data.frame("var.name" = paste0("Var", 1:100),
                    "block" = rep(c(1, 2), each = 50))
dendr2 <- cluster_vars(x = x, block = block)

# The matrix x is first transposed because the function dist calculates
# distances between the rows.
d <- dist(t(x))
dendr3 <- cluster_vars(d = d, method = "single")
```

`hierbase` *hierbase: Enabling Hierarchical Inference*

Description

The hierbase package provides the functions to perform hierarchical inference. The main workflow consists of two function calls.

Functions

The building of the hierarchical tree can be achieved by either of the functions `cluster_vars` or `cluster_positions`. The function `advance_hierarchy` (with 'build-in' group test functions) and `run_hierarchy` (with a user specified group test function) perform the hierarchical testing by going top down through the hierarchical tree and obviously require the hierarchical tree as an input.

References

Meinshausen, N. (2008). Hierarchical testing of variable importance. Biometrika, 95(2), 265-278. Renaux, C., Buzdugan, L., Kalisch, M., and Bühlmann, P. (2020). Hierarchical inference for genome-wide association studies: a view on methodology with software. Computational Statistics, 35(1), 1-40.

`print.hierBase` *Print Object of Class* `hierBase`

Description

Print significant clusters or groups of variables of an object of class `hierBase`.

Usage

```
## S3 method for class 'hierBase'
print(
  x,
  n.terms = 5L,
  digits = max(3, getOption("digits") - 3),
  right = FALSE,
  ...
)
```

Arguments

`x`	an object of class `hierBase`
`n.terms`	maximum number of column names or variables names to be printed per cluster or group of variables.
`digits`	number of significant digits to be used.
`right`	logical value indicating whether the values should or should not be right-aligned.
`...`	additional arguments to `print.data.frame`

Details

The function prints the significant clusters or groups of variables of an object of class `hierBase`. By default, it prints at most the first `n.terms` column or variable names per significant cluster and the number of omitted column names are printed in square brackets (if any).

Value

The returned values is a invisible copy of the object `x`.

See Also

`invisible`.

Examples

```
n <- 200
p <- 100
library(MASS)
set.seed(3)
x <- mvrnorm(n, mu = rep(0, p), Sigma = diag(p))
colnames(x) <- paste0("Var", 1:p)
beta <- rep(0, p)
beta[c(5, 20, 46)] <- 1
y <- x %*% beta + rnorm(n)

dendr <- cluster_vars(x = x)
sign.clusters <- advance_hierarchy(x = x, y = y, dendr = dendr,
                                   test = "F")

# The argument n.terms is useful if there is one or multiple
# significant groups containing many variables.
print(sign.clusters, n.terms = 4)

print(sign.clusters, right = TRUE)

print(sign.clusters, digits = 6)
```

run_hierarchy *Hierarchical Testing*

Description

Hierarchical testing for a given user-specified test function.

Usage

```
run_hierarchy(
  x,
  y,
  dendr,
  test.func,
  clvar = NULL,
  arg.all = list(NULL),
  arg.all.fix = NULL,
  compMOD.same = function(...) NULL,
  compMOD.changing = function(...) NULL,
  alpha = 0.05,
  global.test = TRUE,
  hier.adj = FALSE,
  mt.adj = c("dpBF", "none"),
  agg.method = c("Tippett", "Stouffer"),
  verbose = FALSE,
  sort.parallel = TRUE,
  parallel = c("no", "multicore", "snow"),
  ncpus = 1L,
  cl = NULL
)
```

Arguments

`x`	a matrix or list of matrices for multiple data sets. The matrix or matrices have to be of type numeric and are required to have column names / variable names. The rows and the columns represent the observations and the variables, respectively.
`y`	a vector, a matrix with one column, or list of the aforementioned objects for multiple data sets. The vector, vectors, matrix, or matrices have to be of type numeric.
`dendr`	the output of one of the functions `cluster_vars` or `cluster_positions`.
`test.func`	a test function for groups of variables. See the 'Details' and 'Examples' sections.
`clvar`	a matrix or list of matrices of control variables.
`arg.all`	a list with arguments which is passed on to each call of the functions inputted to the arguments `test.func`, `compMOD.same`, and `compMOD.changing`. See the 'Details' section.

`arg.all.fix`	a vector or list with arguments which is passed on to each call of the functions inputted to the arguments `test.func`, `compMOD.same`, and `compMOD.changing`. See the 'Details' section.
`compMOD.same`	a function which is called once at the beginning of the hierarchical testing procedure and its output is passed on to each call of `test.func`. See the 'Details' section.
`compMOD.changing`	a function which is called once at the beginning of the hierarchical testing procedure and its purpose is to serve as a cache or memory if some output does not have to be recalculated for multiple group tests but the cache can be updated while testing. See the 'Details' section.
`alpha`	the significant level at which the FWER is controlled.
`global.test`	a logical value indicating whether the global test should be performed.
`hier.adj`	a logical value indicating whether the p-values can only increase when going down some given branch. Strong FWER control holds as well if the argument is set to FALSE which is the default option.
`mt.adj`	type of multiple testing correction to be used; either `"dpBF"` (depth-wise Bonferroni multiple adjustment) or `"none"` (no adjustment). See the 'Details' section.
`agg.method`	a character string naming an aggregation method which aggregates the p-values over the different data sets for a given cluster; either `"Tippett"` (Tippett's rule) or `"Stouffer"` (Stouffer's rule). This argument is only relevant if multiple data sets are specified in the function call.
`verbose`	a logical value indicating whether the progress of the computation should be printed in the console.
`sort.parallel`	a logical indicating whether the blocks should be sorted with respect to the size of the block. This can reduce the run time for parallel computation.
`parallel`	type of parallel computation to be used. See the 'Details' section.
`ncpus`	number of processes to be run in parallel.
`cl`	an optional **parallel** or **snow** cluster used if `parallel = "snow"`. If not supplied, a cluster on the local machine is created.

Details

Hierarchical testing is performed by going top down through the hierarchical tree. Testing in some branch only continues if at least one child of a given cluster is significant. The function `run_hierarchy` requires the output of one of the functions `cluster_vars` or `cluster_positions` as an input (argument `dendr`).

The user can specify which hierarchical multiple testing adjustment for the hierarchical procedure is applied. The default is `"dpBF"` (depth-wise Bonferroni multiple adjustment). Alternatively, the user can choose `"none"` (no adjustment). The hierarchical multiple testing adjustment `"dpBF"` guarantees strong family-wise error control if the group test, which is applied for testing a given group, controls the type I error.

The group test function has to be specified by the user; see argument `test.func`. It is required to have the following arguments: `x,y,clvar,colnames.cluster,arg.all,arg.all.fix,mod.large,mod.small`. Although it is fine if not all of them are used inside the function. Most of the arguments are described in the list of arguments above for the function `run_hierarchy` since they are as well specified by the user. The argument `colnames.cluster` contains the column names of the group of variables which should be tested. The arguments `mod.large` and `mod.small` are used as a cache or memory since one can often reuse, say, one initial lasso fit for all the subsequent group tests. The argument `mod.large` corresponds to some result of some initial calculation which is reused later. The argument `mod.small` can be updated continuously through the hierarchical sequential testing procedure. The test function is required to output or return a list with two elements, i.e. the p-value ("pval") calculated for the given group and some object ("mod.small"), which serves as a cache or memory and is passed on to the next function calls in the same branch; see example below. The latter can just be `NULL` if this feature is not used. Compare as well with the above arguments `compMOD.same` and `compMOD.changing` for the function `run_hierarchy`, and see the example below.

If the argument `block` was supplied for the building of the hierarchical tree (i.e. in the function call of either `cluster_vars` or `cluster_positions`), i.e. the second level of the hierarchical tree was given, the hierarchical testing step can be run in parallel across the different blocks by specifying the arguments `parallel` and `ncpus`. There is an optional argument `cl` if `parallel = "snow"`. There are three possibilities to set the argument `parallel`: `parallel = "no"` for serial evaluation (default), `parallel = "multicore"` for parallel evaluation using forking, and `parallel = "snow"` for parallel evaluation using a parallel socket cluster. It is recommended to select `RNGkind("L'Ecuyer-CMRG")` and set a seed to ensure that the parallel computing of the package `hierbase` is reproducible. This way each processor gets a different substream of the pseudo random number generator stream which makes the results reproducible if the arguments (as `sort.parallel` and `ncpus`) remain unchanged. See the vignette or the reference for more details.

Note that if Tippett's aggregation method is applied for multiple data sets, then very small p-values are set to machine precision. This is due to rounding in floating point arithmetic.

Value

The returned value is an object of class `"hierBase"`, consisting a data.frame with the result of the hierarchical testing.

The data.frame has the following columns:

`block`	`NA` or the name of the block if the significant cluster is a subcluster of the block or is the block itself.
`p.value`	The p-value of the significant cluster.
`significant.cluster`	The column names of the members of the significant cluster.

There is a `print` method for this class; see `print.hierBase`.

References

Meinshausen, N. (2008). Hierarchical testing of variable importance. Biometrika, 95(2), 265-278. Renaux, C., Buzdugan, L., Kalisch, M., and Bühlmann, P. (2020). Hierarchical inference for genome-wide association studies: a view on methodology with software. Computational Statistics, 35(1), 1-40.

See Also

`cluster_vars`, `cluster_positions`, and `advance_hierarchy`.

Examples

```
# low-dimensional example with user specified test function
n <- 200
p <- 100
library(MASS)
set.seed(3)
x <- mvrnorm(n, mu = rep(0, p), Sigma = diag(p))
colnames(x) <- paste0("Var", 1:p)
beta <- rep(0, p)
beta[c(5, 20, 46)] <- 1
y <- x %*% beta + rnorm(n)

dendr1 <- cluster_vars(x = x)

# Define own test function: partial F-test
# Many of the arguments of the function below might not be
# used but the function is required to have those arguments.
my.test <- function(x, y, clvar, colnames.cluster, arg.all,
                    arg.all.fix, mod.large, mod.small) {

                       # generate design matrices
                       data.large <- cbind(clvar, x)
                       setdiff.cluster <- setdiff(colnames(x), colnames.cluster)
                       data.small <- cbind(clvar, x[, setdiff.cluster])
                       # Replace data.small if set of indices setdiff.cluster is empty.
                       if (ncol(data.small) == 0) {data.small <- rep(1, length(y))}

                       # compare the models using a partial F test
                       pval <- anova(lm(y ~ data.small),
                                     lm(y ~ data.large),
                                     test = "F")$P[2]

                       return(list("pval" = pval, "mod.small" = NULL))
                    }

# run hierarchical testing
set.seed(76)
sign.clusters1 <- run_hierarchy(x = x, y = y, dendr = dendr1,
                                test.func = my.test)

## With block
# I.e. second level of the hierarchical tree is specified by
# the user. This would allow to run the code in parallel; see the 'Details'
# section.
# The column names of the data frame block are optional.
block <- data.frame("var.name" = paste0("Var", 1:p),
                    "block" = rep(c(1, 2), each = p/2))
dendr2 <- cluster_vars(x = x, block = block)
set.seed(76)
sign.clusters2 <- run_hierarchy(x = x, y = y, dendr = dendr2,
                                test.func = my.test)
```

```
# Access part of the return object or result
sign.clusters2[, "block"]
sign.clusters2[, "p.value"]
# Column names or variable names of the significant cluster in the first row.
sign.clusters2[[1, "significant.cluster"]]
```

Index

advance_hierarchy, 2, *3*, *7*, *9*, *10*, *14*
anova, *3*

cluster_positions, *2–4*, 5, *9*, *10*, *12–14*
cluster_vars, *2–4*, *7*, 7, *10*, *12–14*
cor, *8*

dendrogram, *7*, *9*
dist, *8*

hclust, *8*
hierbase, 10

invisible, *11*

lasso.proj, *3*

print.data.frame, *11*
print.hierBase, *4*, 10, *14*

QF, *3*

RNGkind, *4*, *6*, *9*, *14*
run_hierarchy, *4*, *7*, *9*, *10*, 12, *13*