# Introduction to the R package `hierbase`: Enabling Hierarchical Multiple Testing

Claude Renaux and Peter Bühlmann
Seminar for Statistics, ETH Zürich

November 8, 2021

## 1 Cite `hierbase`

If you use the `hierbase` package, please cite the papers Meinshausen, N. (2008). Hierarchical testing of variable importance. *Biometrika*, 95(2), 265-278 and 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.

## 2 Introduction

A major goal in high-dimensional statistics is to assign statistical significance of single covariates for a response of interest. Especially the inference part in terms of statistical significance testing (or confidence intervals) is, however, often overly ambitious and requires stringent assumptions on the well-posedness of the design matrix of the covariates in high dimensions. Instead, we advocate to use hierarchical inference, as proposed earlier with the same motivation by Meinshausen (2008) and further extended to simultaneously analyse multiple data sets by Renaux et al. (2020). It is a data-driven multiple testing approach to find significant groups or possibly singletons of covariates. The advantage is that groups of covariates are typically much easier to identify and if the signal is sufficiently strong relative to the correlation structure, our method is still able to detect significant single covariates. The procedure goes top-down through a hierarchical tree from larger to smaller groups and tries to find an as fine resolution of significant groups as possible.

Since many tests are done in a sequential structure given by the hierarchical tree, a multiple testing adjustment has to be applied. Meinshausen (2008) first proposed a hierarchical multiple testing adjustment, which we call depth-wise Bonferroni: it (essentially) performs Bonferroni adjustment with respect to multiplicity of the number of tests at each depth of the hierarchical tree.

## 3 Hierarchical Cluster Tree

The tree has to partition the covariates in non-overlapping groups for each height of the tree. The partitions are coarser in the top part and get finer toward the bottom of the tree. The top node in the tree is always the entire group of all covariates and corresponds to the global null-hypothesis. Our R package includes two functions for building a hierarchical tree. `cluster_vars` or `cluster_positions`.

A hierarchical tree can be built using hierarchical clustering of the $p$ covariates using $1 - (\text{Person's correlation})^2$ as a dissimilarity measure and average linkage. This is performed by default using the function `cluster_vars` but many other options are offered.

If there exists a meaningful ordering of the covariates such that closer covariates are more similar, then the function `cluster_positions` could be used. It builds a hierarchical tree using recursive binary partitioning of consecutive covariates.

The user is free in define a different hierarchical tree as long as it can be stored as a dendrogram in R. See the structure of the output of either `cluster_var` or `cluster_position` in order to use the same structure when calling the hierarchical procedure.

## 4 Toy example for the function `advance_hierarchy`

Our R package `hierbase` implements depth-wise Bonferroni. We demonstrate the function `advance_hierarchy` in a toy example. The following arguments of the function are specified: `x` the data matrix, `y` the response, `dendr` a dendrogram of the hierarchical tree (e.g. the output of the function `cluster_vars`), and `test` a character string representing which of the ready-to-use test functions should be used.

```
# load the packages
library(hierbase)
library(MASS) # for generating x

# random number generator (for parallel computing)
RNGkind("L'Ecuyer-CMRG")

# generate a toy data set
n <- 100
p <- 50 # 200
set.seed(3)                             # set a seed
x <- mvrnorm(n, mu = rep(0, p), Sigma = diag(p))  # data matrix x
colnames(x) <- paste0("Var", 1:p)       # column names
beta <- rep(0, p)                       # coefficients
beta[c(5, 20, 46)] <- 1                 # three active covariates
y <- x %*% beta + rnorm(n)              # response

# estimate hierarchical tree
dendr1 <- cluster_vars(x = x)

# run hierarchical procedure
set.seed(4)
res <- advance_hierarchy(x, y, dendr = dendr1, test = "QF")

## [1] "step is 4"
## [1] "step is 4"
## [1] "step is 4"
## [1] "step is 4"
```

```
## [1] "step is 3"
## [1] "step is 4"
## [1] "step is 3"
## [1] "step is 4"
## [1] "step is 3"
## [1] "step is 4"
## [1] "step is 3"
## [1] "step is 4"
## [1] "step is 4"
## [1] "step is 3"
## [1] "step is 4"
## [1] "step is 4"

res

##   block p.value   significant.cluster
## 1 NA    5.248e-05 Var46
## 2 NA    0.0001057 Var20
## 3 NA    0.0001417 Var5
```

If no second level of the hierarchical tree is specified by the user (see argument `block` of the function `cluster_vars`), then `NA`s are displayed in the column `block` of the above output. See Section 6 for an example how one can define the second level of the hierarchical tree and run the code in parallel.

Note that the printed lines starting with "step is ..." are printed by the test function `QF` used in this example from the R package `SIHR`. The methodology and theory for the test function `QF` can be found in Guo et al. (2020).

## 5 Toy example for the function `run_hierarchy`

Alternatively, one can use the function `run_hierarchy` instead of `advance_hierarchy`. The clustering step remains the same. The function `run_hierarchy` is very generic and can be called with the user's favorite test function. The test function is passed on as an argument of the function `run_hierarchy`.

```
# # estimate hierarchical tree
# dendr1 <- cluster_vars(x = x)

# define test function
# low-dimensional partial F-Test
test.func.F <- function(x, y, clvar, colnames.cluster,
                        arg.all, arg.all.fix, mod.large,
                        mod.small) {
  ## larger model
  data.large <- cbind(clvar, x)
```

```
  # estimate larger model
  mod.large <- lm(y ~ data.large)

  ## smaller model
  setdiff.cluster <- setdiff(colnames(x), colnames.cluster)
  data.small <- cbind(clvar, x[, setdiff.cluster])
  # special case if data.small is empty
  if (ncol(data.small) == 0) {data.small <- rep(1, length(y))}

  # calculate smaller model
  mod.small <- lm(y ~ data.small)

  ## compare the models
  # partial F test
  pval <- anova(mod.small, mod.large, test = "F")$P[2]

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

# run hierarchical procedure
set.seed(4)
res2 <- run_hierarchy(x, y, dendr = dendr1, test.func = test.func.F)
res2
```

## 6 Parallel

All the functions can easily be run in parallel if the second level of the hierarchical tree is specified by the user. Typically, one would specify a partition consisting of, say, five groups which represent the second level of the tree. This is specified as an argument in the function call of `cluster_vars` or `cluster_positions` if desired. The name of those groups would appear in the column block of the output of the hierarchical procedure.

```
## With block
# The user defines the second level of the hierarchical tree.
block <- data.frame("var.name" = paste0("Var", 1:p),
                    "block" = rep(c(1, 2), each = p/2))

# Estimate the hierarchical cluster tree in parallel.
# The argument block defines the second level of the tree.
dendr2 <- cluster_vars(x = x, block = block,
                       # the following arguments have to be specified
                       # for parallel computation
                       parallel = "multicore",
                       ncpus = 2)
```

```r
# Run the hierarchical procedure in parallel.
set.seed(76)
res2 <- advance_hierarchy(x = x, y = y, dendr = dendr2,
                          test = "QF",
                          # the following arguments have to be specified
                          # for parallel computation
                          parallel = "multicore",
                          ncpus = 2)
```

See the help files of the functions `run_hierarchy` or `advance_hierarchy` for more details about how one can run the code in parallel.

Note that it is possible to analyze multiple data sets simultaneously by specifying the different responses and data matrices each as elements of two lists; see the help files for more details. This makes sense if data sets from, say, multiple studies with the same response and similar or same data matrices are analyzed.

## References


Guo, Z., Renaux, C., Bühlmann, P., and Cai, T. T. (2020). Group inference in high dimensions with applications to hierarchical testing. *arXiv preprint arXiv:1909.01503*.

Meinshausen, N. (2008). Hierarchical testing of variable importance. *Biometrika*, 95:265–278.

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.