geecc - searching for associations between gene sets

Markus Boenn

October 13, 2014

1 Introduction

A standard application of gene expression data is to search for genes which are differentially expressed between two (or more) conditions. For this, depending on the nature of the data, tests for continous values (microarrays, t-test and derivatives, limma-package) or tests discrete counts (RNA-Seq, edgeR- [Rob10] or DESeq-package [And10], just to name two) can be applied. Having found a list of differentially expressed genes (DEG), the next step is to get an impression about the functions of these DEG. I.e. scientists want to know what these genes are doing and why are they differentially expressed in their experiment.

For this, searching for enriched gene sets has become standard analysis: The data is implicitly reorganised in 2×2 contingency tables and a hypergeometric or chi-squared test is applied to get a P-value. If the P-value is sufficiently small, the assumption of independence is rejected and some association between the gene sets assumed. The most popular database providing gene sets is the Gene Ontology (GO) `geneontology.org`. Other popular gene set systems are PFam, KEGG or {C|K}OG.

Most tools focus on a simple enrichment analysis checking whether there is a dependency between a given DEG list and a list of gene sets, for instance GOstats [Fal07]. However, Simpson's Paradoxon tells us that unresolved details in the gene sets might put a bias to the results and its interpretation. To tackle this problem, the geecc-package allows for enrichment analysis between three categories. A potential third category of interest might be the sequence length, chromosomal position, GC content or phylostrata.

Basically, geecc permits the application of tests for the hierarchy of log-linear models (e.g. [Agr92]). This includes tests where the assumption of

- mutual independence
- single pairwise (joint) independence
- two pairwise (conditional) independence
- homogenous association, no three-way interaction

defines the null-model (null-hypothesis). Note that the alternative model is the saturated model. Using these tests, questions like 'Are these variables mutually independent from each other?' or 'Is there evidence for three-way interaction?'

can be answered. Depending on the specific question of interest, testing for different directions of enrichment, i.e. over-representation, under-representation or both, is possible. For this, the geecc-package calls routines from the hypergea-package (`hypergeom.test()`) and MASS-package (`loglm()`). The routines from hypergea-package are calculating P-values from a hypergeometric distribution, whereby this calculation is done with parallelized C code. The routines can be used if highly accurate P-values are required and approximation by chi-squared tests may not be reliable. The `loglm()`-function provides approximate chi-squared tests via convenient access by formulas for linear models. Tables 1 and 2 give an overview for different null-models, possible test-directions and test-types.

Table 1: **Tests for 2×2 questions.**

null-model	direction	test
mutual independence	one-sided	hypergeometric
mutual independence	two-sided	hypergeometric, χ^2

Table 2: **Tests for $2 \times 2 \times 2$ questions.**

null-model	direction	test
mutual independence	one-sided	hypergeometric
mutual independence	two-sided	hypergeometric, χ^2
joint independence	one-sided	hypergeometric
joint independence	two-sided	hypergeometric, χ^2
conditional independence	two-sided	χ^2
homogenous association	two-sided	χ^2

2 General workflow - a worked example

The workflow can be divided into four distinct steps. In a first step the user have to prepare the data, i.e. a decision has to be made what are the differentially expressed genes (proper threshoulds for P-values and fold-changes), which GO terms should be considered, and, in case that a third category should be taken into account, assign the gene ids to proper categories. The outcome of this step should be a named list, where each item is again a named list. This inner list contains the variables and the items assigned to them.

In this worked example we introduce the different steps that have to be done during a geecc analysis. We use gene expression data from Marioni *et al.* [Mar08] (downloaded from `http://giladlab.uchicago.edu/data.html`), who compared differences in gene expression in liver and kidney.

To speed up the examples in the vignette we consider only the first 15000 probe sets.

2.1 Step 1: Data preparation

```
> library(geecc)
> #load marioni data set
> data(marioni)
> marioni <- marioni[1:15000, ]
> # adjust for multiple testing and get probe sets which are
> # at least two-fold regulated and fdr smaller than 5 %
> fdr <- p.adjust(marioni[, "Pvalue"], "fdr")
> deg.diff <- rownames(marioni)[ which(fdr < 0.05) ]
> deg.up <- rownames(marioni)[ which(fdr < 0.05 & marioni[, "logFC"] > 0) ]
> deg.down <- rownames(marioni)[ which(fdr < 0.05 & marioni[, "logFC"] < 0) ]
> sapply(list(deg.diff, deg.up, deg.down), length)

[1] 9983 5337 4646
```

Next, prepare the gene sets.

```
> library(GO.db)
> library(hgu133plus2.db)
> ## divide sequence lengths into 33 percent quantiles
> seqlen <- setNames(marioni[, "End"] - marioni[, "Start"] + 1, rownames(marioni))
> step <- 33; QNTL <- seq(0, 100, step)
> qntl <- cbind(quantile(seqlen, prob=QNTL/100), QNTL)
> cc <- cut(seqlen, breaks=qntl[,1], labels=qntl[-length(QNTL),2], include.lowest=TRUE)
> seqlen.qntl <- cbind(seqlen, cc )
> #check if there are three groups of same size
> table(seqlen.qntl[,2])

   1    2    3
4950 4950 4950

> ## prepare a list of levels for each category
> ## restrict to GO category 'cellular component' (CC)
> category1 <- list( diff=deg.diff, up=deg.up, down=deg.down )
> category2 <- GO2list(db=hgu133plus2GO2PROBE, go.cat="CC")
> category3 <- split(rownames(seqlen.qntl), factor(seqlen.qntl[,2]))
> names(category3) <- as.character(c(QNTL[1:(length(QNTL)-1)]))
> ## check content of each category list
> lapply(category1[1:3], head)

$diff
[1] "205626_s_at" "220281_at"   "207102_at"   "205978_at"   "206345_s_at"
[6] "219630_at"

$up
[1] "205626_s_at" "220281_at"   "205978_at"   "219630_at"   "205799_s_at"
[6] "224179_s_at"

$down
[1] "207102_at"   "206345_s_at" "207584_at"   "220383_at"   "206354_at"
[6] "206386_at"
```

```
> lapply(category2[1:3], head)

$`GO:0000015`
          IEA           IEA           IEA           IEA           IEA
"201231_s_at" "216554_s_at" "217294_s_at"   "240258_at"   "201313_at"
          IEA
  "204483_at"

$`GO:0000109`
           IDA            IDA            IDA            IDA            IDA
  "1554882_at" "1554883_a_at"    "205162_at"    "203719_at"  "203720_s_at"
           IDA
   "228131_at"

$`GO:0000110`
        IDA         IDA
"210158_at" "235215_at"

> lapply(category3[1:3], head)

$`0`
[1] "206386_at"   "224179_s_at" "207262_at"   "236646_at"   "206350_at"
[6] "242601_at"

$`33`
[1] "219630_at" "205755_at" "206753_at" "205576_at" "206505_at" "207096_at"

$`66`
[1] "205626_s_at" "220281_at"   "207102_at"   "205978_at"   "206345_s_at"
[6] "207584_at"

> CatList <- list(deg=category1, go=category2, len=category3)
```

2.2 Step 2: Initialize objects

In a second step, the `concubfilter`-object (storing the filters that should be applied during analysis) and the `concub`-object (storing categories, their variables and other options for the categories) have to be initialized.

In this worked example we perform a simple GO enrichment analysis for differentially expressed genes. We don't want to consider only over-represented GO terms, but also under-represented ones, so we apply a `"two.sided"` test.

```
> ## run a simple two-way analysis on 'deg' and 'go'
> ## create a ConCubFilter-object
> CCF.obj <- new("concubfilter", names=names(CatList)[1:2], p.value=0.5,
+   test.direction="two.sided", skip.min.obs=2)
> ## create a ConCub-object
> CC.obj <- new("concub", fact=CatList[1:2], population=rownames(marioni),
+   approx=5, null.model=~deg+go)
```

2.3 Step 3: Run tests and filter results

In the third step we run the test and apply some additional filtering on the outcome.

In this example we consider only the first 400 GO terms.

```
> ## check current filter settings and change some filters
> CCF.obj

####################
# current filter settings
####################
Number of categories: 2
Maximum P-value: 0.1
Minimum absolute log2 odds ratio: 0
Test direction: two.sided

Skip test in case of
        no items: TRUE
        number of items less than: 1
        minimum marginal (to skip small gene sets): deg=0,go=0

Layers to be dropped in case of
        insignificant P-values: deg=FALSE,go=FALSE
        wrong direction: deg=FALSE,go=FALSE
        small abs(log2(or)): deg=FALSE,go=FALSE
####################

> drop.insignif.layer(CCF.obj) <- setNames(c(FALSE, TRUE), names(CatList)[1:2])
> p.value(CCF.obj) <- 0.01
> CCF.obj

####################
# current filter settings
####################
Number of categories: 2
Maximum P-value: 0.01
Minimum absolute log2 odds ratio: 0
Test direction: two.sided

Skip test in case of
        no items: TRUE
        number of items less than: 1
        minimum marginal (to skip small gene sets): deg=0,go=0

Layers to be dropped in case of
        insignificant P-values: deg=FALSE,go=TRUE
        wrong direction: deg=FALSE,go=FALSE
        small abs(log2(or)): deg=FALSE,go=FALSE
####################

> CC.obj3 <- filterConCub(obj=CC.obj2, filter=CCF.obj, p.adjust.method="BH")
```

Additional filters can be applied, which drop useless test results, for instance those with insignificant P-values (according to the setting of `p.value` in the `concubfilter`-object). In addition, we are able to adjust the P-values for multiple testing at this stage. This is done within the `filterConCub`-function by calling the R-routine `p.adjust` with the '`method`' set by the user (Benjamini-Hochberg adjustment [BH95] in this example).

2.4 Step 4: Visualization and storing final table

In the last step, the test results are visualized and can be stored in a `data.frame` for further work. Results are visualized in a simple heatmap. This fosters interpretation of possible associations in the data. The `heatmap.2`-function from gplots-package is used. Arguments of this function (see man page of `heatmap.2`), especially size of labels and clustering options, can be adapted to the users needs by passing the settings via the '`args_heatmap.2`'-parameter (a list).

```
> ## interpretation of raw GO ids is difficult, use term description
> translation <- list(go=setNames(sapply(names(category2), Term), names(category2)))
> ## pdf("output.2w.pdf")
> plotConCub( obj=CC.obj3, filter=CCF.obj, col=list(range=c(-5,5))
+   , alt.names=translation, t=TRUE, dontshow=list(deg=c("diff"))
+   , args_heatmap.2=list(Rowv=TRUE, dendrogram="row", margins=c(3,12))
+ )
> ## dev.off()
> res2w <- getTable(obj=CC.obj3, na.rm=TRUE)
```

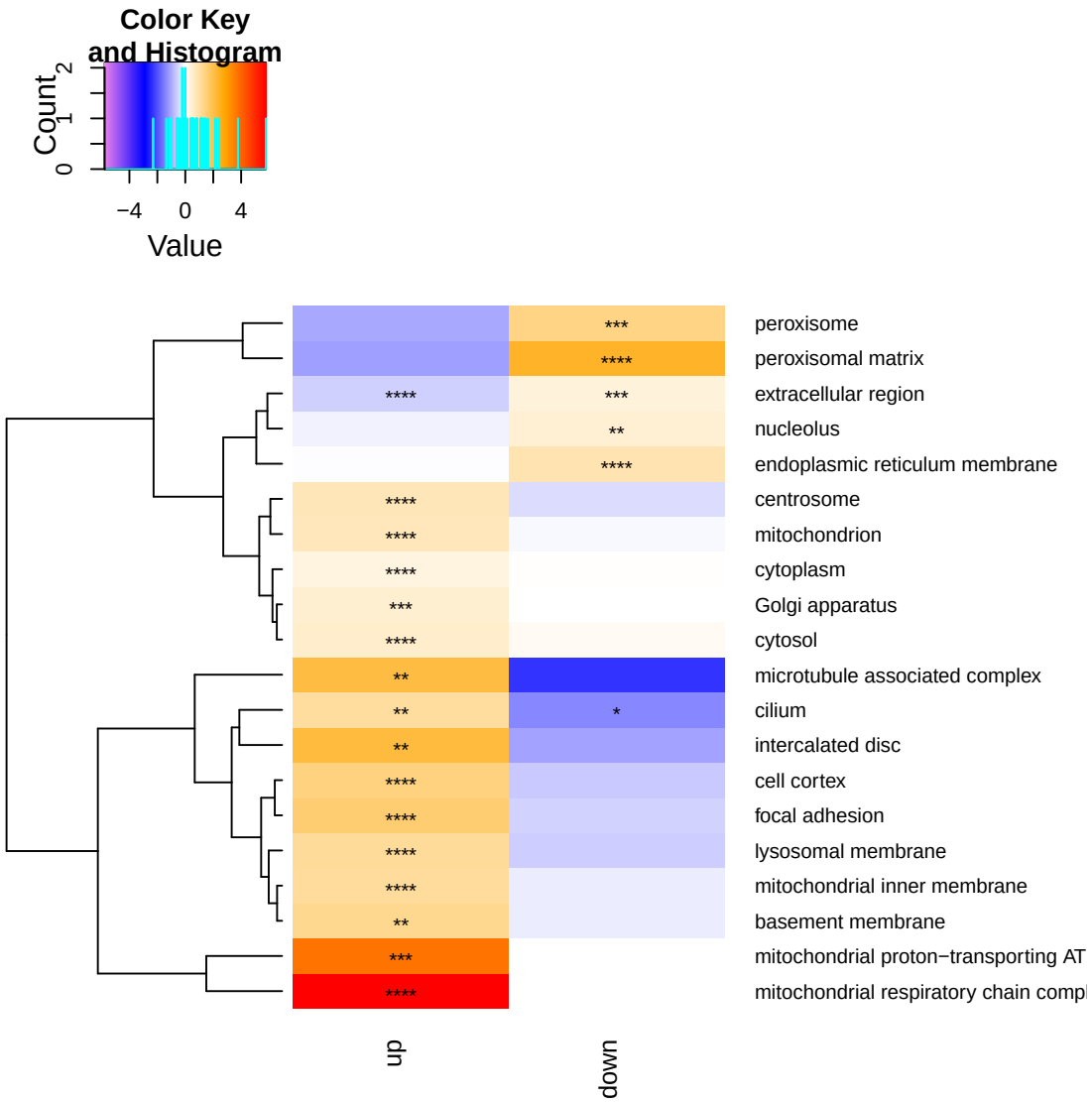

Figure 1: **Outcome of step 4.** Heatmap showing the result of a simple two way analysis with a category with two variables **deg** (*up* and *down*) and a category with many variables (**go**). Colored cells are the $\log_2$ odds ratio indicating over- and under-representation. Stars show significance of the odds ratio.

For this example the outcome is shown in Figure 1. Each cell in the heatmap corresponds to the ($\log_2$) odds ratio for the variables compared. The default is to express enrichment (over-representation) of variables by red colors. Under-representation is emphasized by blue colors. Stars (or a dot) within each cell show the degree of significance of over- and under-representation in analogy to other tests, i.e. '.' corresponds to a P-value between 0.05 and 0.01, whereas '****' indicates that the P-value is smaller than 0.001, for example. Some cells are white colored. These cells usually correspond to not-performed test due to filter-settings.

In three-ways approaches the `plotConCub`-function will create multiple heatmaps at the same time. So it is recommended to surround this command by a loop which puts each heatmap into a separate (svg, jpg, ...) file (can be controled by the '`fix.cat`'-parameter) or to put everything into a pdf-file (as indicated by the comment out lines in the example).

The `filterConCub`-function allows to adjust P-values for multiple testing at an early stage. However, the adjustment is made for the whole data, which might not be the best time point for adjustment for some experiments. In this case, setting `p.adjust.method="none"` skips P-value adjustment. The adjustment can then be done on the table obtained by the `getTable`-function:

```
> res2wa <- getTable(CC.obj3, na.rm=TRUE, dontshow=list(deg=c("up", "down")))
> res2wa[, 'p.value.bh'] <- p.adjust(res2wa[, 'p.value'], method="BH")
> res2wa <- res2wa[ res2wa[, 'p.value.bh'] <= 0.05, ]
```

Here we select the results for all differentially expressed genes and apply a Benjamini-Hochberg adjustment on the P-values.

3 Working with three categories

3.1 Testing for mutual independence

We stay with the data from the worked example. A typical first question that might be asked is, if independence of **deg**, **len** and **go** for some combinations of variables can be confirmed.

We confine ourselves to the results from the worked example to test for mutual independence of the three categories. For this, we have to set the '`null.model`'-parameter accordingly: `null.model=`$\sim$ `deg+go+len`.

```
> CCF.obj.3wmi <- new("concubfilter", names=names(CatList), p.value=0.5,
+   test.direction="two.sided", skip.min.obs=2)
> CC.obj.3wmi <- new("concub", fact=CatList, population=rownames(marioni),
+   null.model=~deg+go+len)
> gorange <- as.character(unique(res2w$go))
> CC.obj2.3wmi <- runConCub( obj=CC.obj.3wmi, filter=CCF.obj.3wmi,
+   nthreads=4, rng=list(go=gorange) )
> drop.insignif.layer(CCF.obj.3wmi) <- setNames(c(FALSE, TRUE, FALSE), names(CatList))
> p.value(CCF.obj.3wmi) <- 0.01
> CC.obj3.3wmi <- filterConCub( obj=CC.obj2.3wmi, filter=CCF.obj.3wmi,
+   p.adjust.method="BH")
```

```
> ## pdf("output.3w.pdf")
> plotConCub( obj=CC.obj3.3wmi, filter=CCF.obj.3wmi, col=list(range=c(-5,5))
+   , alt.names=translation, t=FALSE, dontshow=list(deg=c("diff"))
+   , args_heatmap.2=list(Rowv=TRUE, dendrogram="row", margins=c(3,12))
+ )
> ## dev.off()
> res3wmi <- getTable(obj=CC.obj3.3wmi, na.rm=TRUE)
```

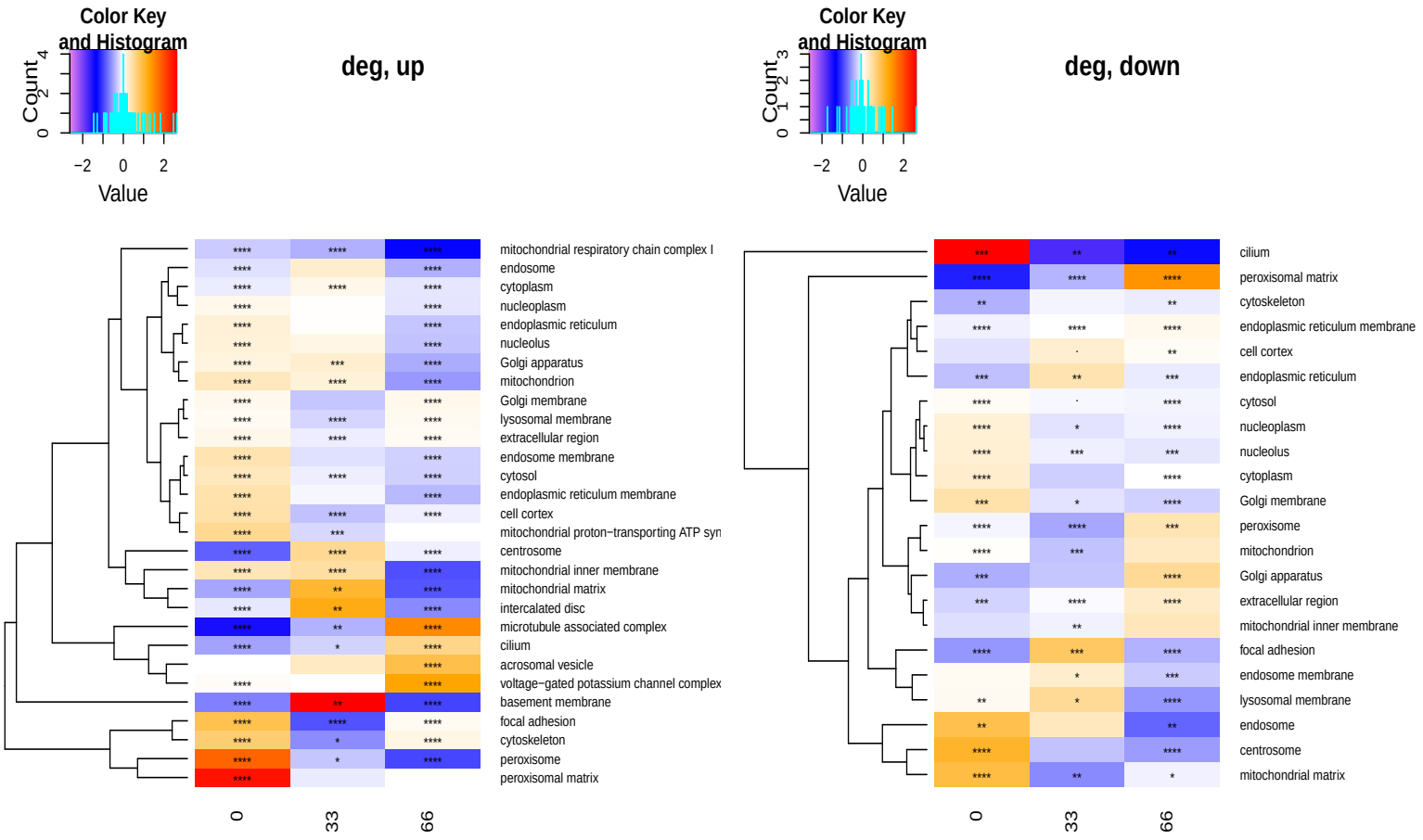

Figure 2: **Outcome of a test on mutual independence.** Results for up- and down-regulated genes.

We see some differences in the sequence lengths of up- and down-regulated genes. For instance, we see that enrichment of GO term 'peroxisomal matrix' in up-regulated genes is caused by small-size sequences, whereas enrichment of GO term 'acrosomal vesicle' is caused by long sequences.

3.2 Joint independence

Testing for mutual independence of all three categories gives a first idea, but finally little information since there are multiple non-three-way associations that can cause rejection of this nullhypothesis. We can go deeper to figure out the single pairwise (joint) dependencies applying $\sim$ `deg+go*len`, $\sim$ `deg*go+len`, or $\sim$ `deg*len+go`.

```
> CCF.obj.3wsp1 <- new("concubfilter", names=names(CatList), p.value=0.5,
+   test.direction="two.sided", skip.min.obs=2)
> CC.obj.3wsp1 <- new("concub", fact=CatList, population=rownames(marioni),
+   null.model=~deg+go*len)
> gorange <- as.character(unique(res3wmi$go))
> CC.obj2.3wsp1 <- runConCub( obj=CC.obj.3wsp1, filter=CCF.obj.3wsp1, nthreads=2,
+   rng=list(go=gorange) )
> drop.insignif.layer(CCF.obj.3wsp1) <- setNames(c(FALSE, TRUE, FALSE), names(CatList))
> p.value(CCF.obj.3wsp1) <- 0.05
> CC.obj3.3wsp1 <- filterConCub( obj=CC.obj2.3wsp1, filter=CCF.obj.3wsp1,
+   p.adjust.method="BH")
```

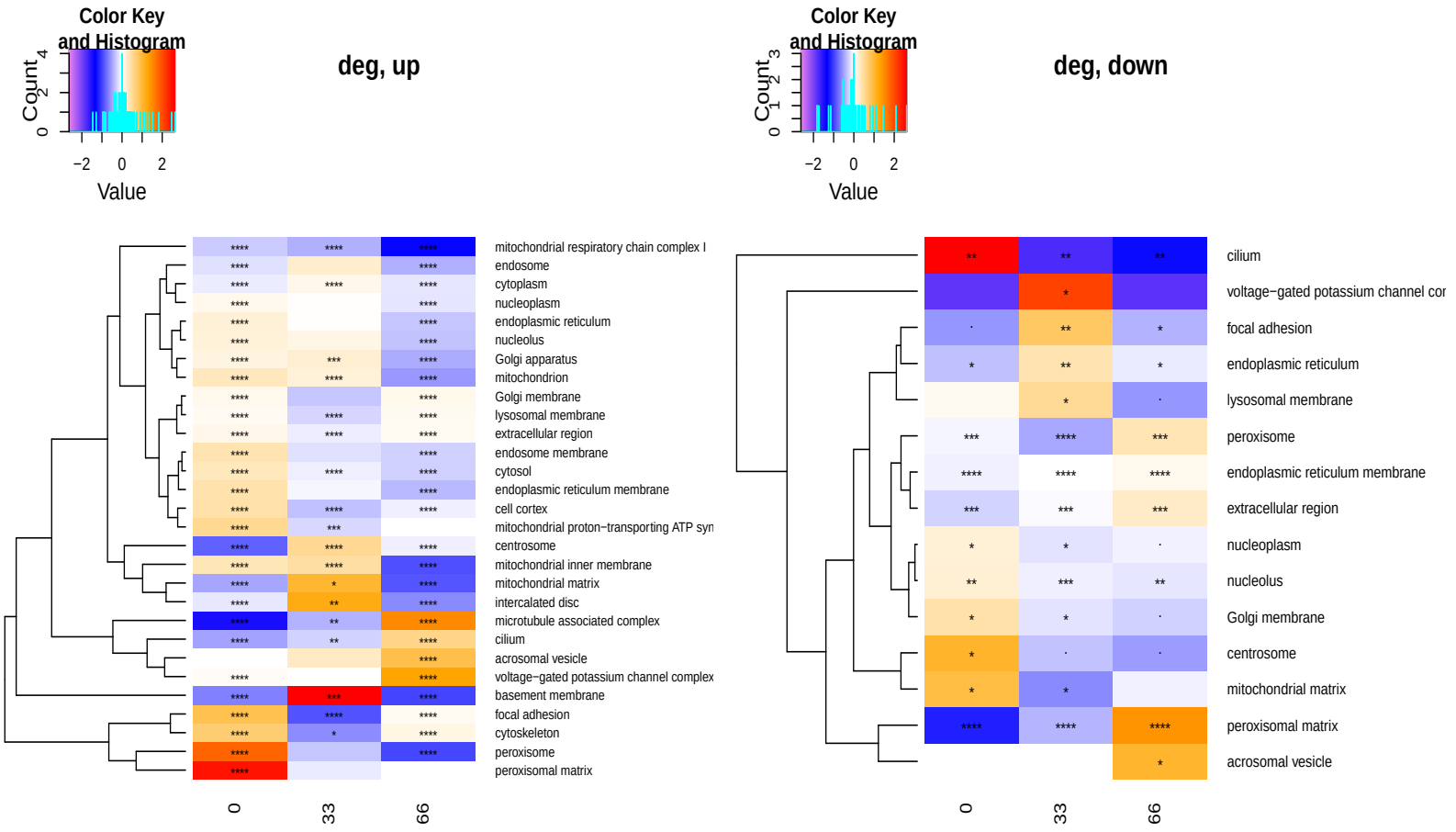

Figure 3: **Outcome of a test on joint independence.** Results for up- and down-regulated genes.

From Figure 3 we see that this null-modell does not sufficiently describe most of the combinations between variables of categories. There are still many rejections (significant P-values). For the following variables this null-model seem to fit:

```
> res3wsp1 <- getTable(obj=CC.obj3.3wsp1, na.rm=TRUE)
> res3wmi_sig <- res3wmi[res3wmi$p.value < 0.05, ]
> res3wsp1_sig <- res3wsp1[res3wsp1$p.value < 0.05, ]
> head(res3wmi_sig[ !(do.call("paste", res3wmi_sig[,names(CatList)])
+   %in% do.call("paste", res3wsp1_sig[,names(CatList)])), 1:8])

    deg         go len n.deg  n.go n.len n.tags      p.value
21 down GO:0005737   0  4646 10498  4950    301 0.000000e+00
24 down GO:0005739   0  4646  2479  4950     71 1.782605e-05
39 down GO:0005765   0  4646   606  4950     11 7.214359e-03
42 down GO:0005768   0  4646   395  4950     14 4.949656e-03
57 down GO:0005794   0  4646  1700  4950     38 1.485809e-04
63 down GO:0005829   0  4646  6815  4950    159 0.000000e+00
```

So, for these variables there seem to be no hint that differential expression on the one hand and function and sequence length on the other hand are not independent from each other.

3.3 No threeway interaction vs. saturated model

Finally, we test for the last null-model in the hierarchy of log-linear models. Here we detect those combinations which are probably really completely depending on each other.

```
> CCF.obj.3wnti <- new("concubfilter", names=names(CatList), p.value=0.5,
+   test.direction="two.sided", skip.min.obs=2)
> CC.obj.3wnti <- new("concub", fact=CatList, population=rownames(marioni),
+   null.model=~len*go+deg*go+deg*len)
> gorange <- as.character(unique(res3wmi$go))
> CC.obj2.3wnti <- runConCub( obj=CC.obj.3wnti, filter=CCF.obj.3wnti,
+   nthreads=4, rng=list(go=gorange) )
> CC.obj3.3wnti <- filterConCub( obj=CC.obj2.3wnti, filter=CCF.obj.3wnti,
+   p.adjust.method="BH")

> ## pdf("output.3w.ha.pdf")
> plotConCub( obj=CC.obj3.3wnti, filter=CCF.obj.3wnti, col=list(range=c(-5,5))
+   , alt.names=translation, t=FALSE, dontshow=list(deg=c("diff"))
+   , args_heatmap.2=list(Rowv=TRUE, dendrogram="row", margins=c(3,12))
+ )
> ## dev.off()
```

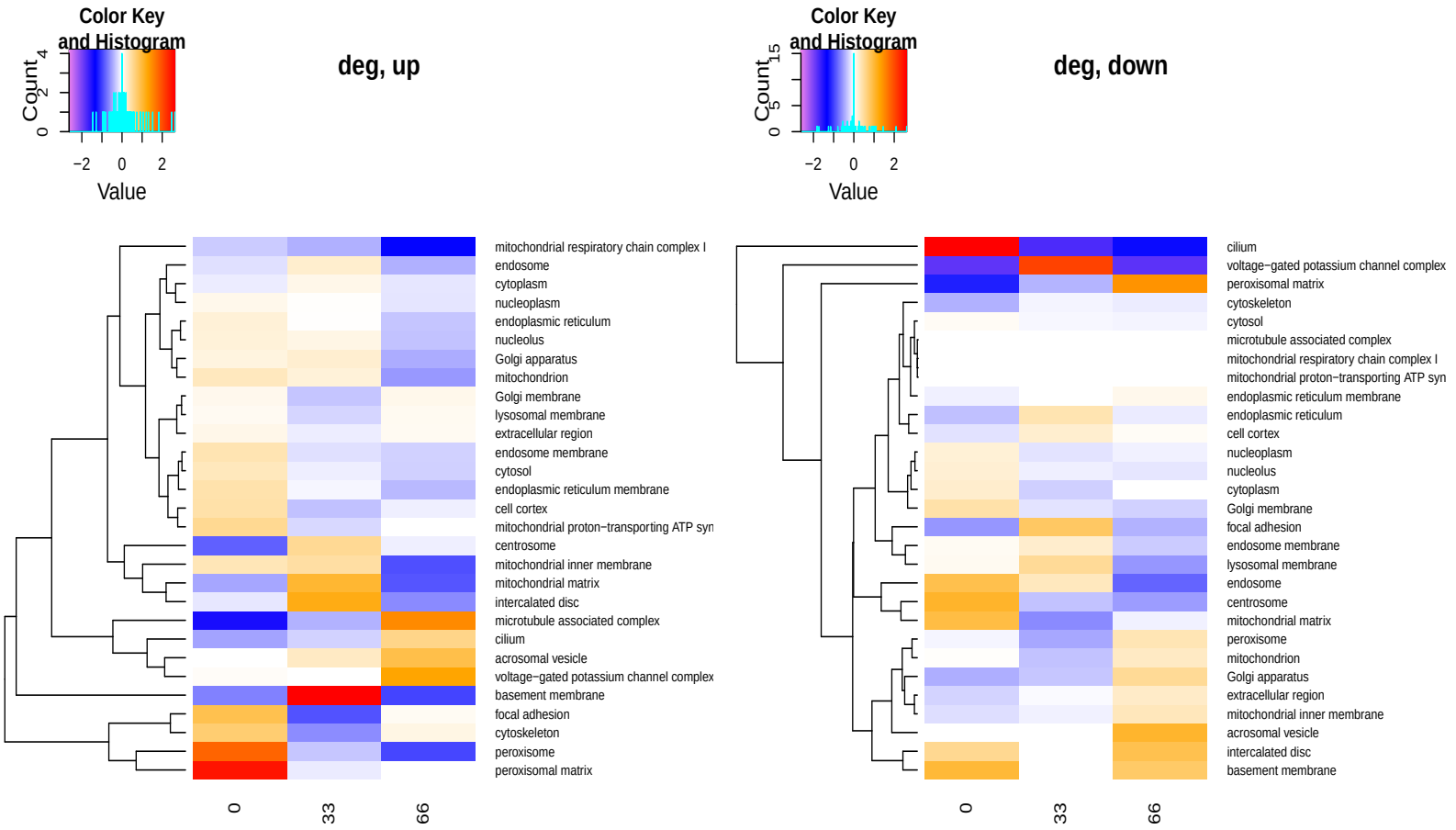

Figure 4: **Outcome of a test on no three-way interaction.** Up- and down-regulated genes.

It seems that this model describes the data well, i.e. for all combinations of variables considered here there seem to be no evidence that assumption of no three-way interaction should be rejected. However, we skipped several other null-models in the hierarchy of log-linear models, which might fit the data as well and in a more parsimonious way.

3.4 Making use of ordinal variables

In general, the category variables are nominal, but not ordinal. However, in the example above the third category (sequence length **len**) is ordinal. `geecc` provides additional methods to ask for simple extended questions. For instance, one might not be interested in enrichment of up-regulated genes having the lowest sequence length in the binning and which also belong to a certain GO term. Instead, one might want to ask, if there is a relationship for 'short' ('0') sequences without knowing when a sequence is a short one and when it is not short anymore. In such case one might make use of the additional '`options`'. For the case described in this paragraph, one might set `options=list(len=list(grouping="cumf"))`. This cumulatively summarizes the **len**-levels from left to right and performs the test.

References

[Agr92] Alan Agresti. A survey of exact inference for contingency tables. *Statistical Science*, pages 131–153, 1992.

[And10] Anders, S. and Huber, W. Differential expression analysis for sequence count data. *Genome Biol*, 11(10):R106, 2010.

[BH95] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 289–300, 1995.

[Fal07] Falcon, S. and Gentleman, R. Using GOstats to test gene lists for GO term association. *Bioinformatics*, 23(2):257–258, 2007.

[Mar08] Marioni, John C and Mason, Christopher E and Mane, Shrikant M and Stephens, Matthew and Gilad, Yoav. RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays. *Genome research*, 18(9):1509–1517, 2008.

[Rob10] Robinson, Mark D and McCarthy, Davis J and Smyth, Gordon K. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, 2010.