OUTRIDER - OUTlier in RNA-Seq fInDER

Felix Brechtmann¹, Christian Mertes¹, Agne Matuseviciute¹, Vicente Yepez¹,², Julien Gagneur¹,²

¹ Technical University Munich, Department of Informatics, Munich, Germany
² Quantitative Biosciences Munich, Gene Center, Ludwig-Maximilians Universität München, Munich, Germany

October 24, 2023

Abstract

In the field of diagnostics of rare diseases, RNA-seq is emerging as an important and complementary tool for whole exome and whole genome sequencing. OUTRIDER is a framework that detects aberrant gene expression within a group of samples. It uses the negative binomial distribution which is fitted for each gene over all samples. We additionally provide an autoencoder, which automatically controls for co-variation before fitting. After fitting, each sample can be tested for aberrantly expressed genes. Furthermore, OUTRIDER provides functionality to easily filter unexpressed genes, to analyse the data as well as to visualize the results.

If you use OUTRIDER in published research, please cite:

# Contents

1 Introduction ......................................................... 3
2 Prerequisites .......................................................... 4
3 A quick tour ............................................................ 4
4 An *OUTRIDER* analysis in detail ................................. 6
   4.1 OutriderDataSet ................................................. 7
   4.2 Preprocessing .................................................... 7
   4.3 Controlling for Confounders ................................. 10
   4.4 Finding the right encoding dimension \( q \) .............. 13
      4.4.1 Excluding samples from the autoencoder fit ........ 13
   4.5 Fitting the negative binomial model ..................... 14
   4.6 P-value calculation ........................................... 14
   4.7 Z-score calculation ........................................... 15
5 Results ................................................................. 15
   5.1 Results table ................................................... 15
   5.2 Number of aberrant genes per sample .................... 17
   5.3 Volcano plots .................................................. 18
   5.4 Gene level plots ............................................... 19
6 Additional features .................................................. 21
   6.1 Using PEER to control for confounders ................. 21
   6.2 Power analysis ................................................ 23
References ............................................................. 23
OUTRIDER (OUTlier in RNA-seq fInDER) is a tool for finding aberrantly expressed genes in RNA-seq samples. It does so by fitting a negative binomial model to RNA-seq read counts, correcting for variations in sequencing depth and apparent co-variations across samples. Read counts that significantly deviate from the distribution are detected as outliers. OUTRIDER makes use of an autoencoder to control automatically for confounders within the data. A scheme of this approach is given in Figure 1.

OUTRIDER context-dependent outlier detection

---

Figure 1: Context-dependent outlier detection. The algorithm identifies gene expression outliers whose read counts are significantly aberrant given the co-variations typically observed across genes in an RNA sequencing data set. This is illustrated by a read count (left panel, fifth column, second row from the bottom) that is exceptionally high in the context of correlated samples (left six samples) but not in absolute terms for this given gene. To capture commonly seen biological and technical contexts, an autoencoder models co-variations in an unsupervised fashion and predicts read count expectations. By comparing the earlier mentioned read count with these context-dependent expectations, it is revealed as exceptionally high (right panel). The lower panels illustrate the distribution of read counts before and after applying the correction for the relevant gene. The red dotted lines depict significance cutoffs.

Differential gene expression analysis from RNA-seq data is well-established. The packages DESeq2[1] or edgeR[2] provide effective workflows and preprocessing steps to perform differential gene expression analysis. However, these methods aim at detecting significant differences between groups of samples. In contrast, OUTRIDER aims at detecting outliers within a given population. A scheme of this difference is given in figure 2.
Figure 2: Scheme of workflow differences. Differences between differential gene expression analysis and outlier detection.

2 Prerequisites

To get started on the preprocessing step, we recommend to read the introductions of DESeq2\cite{1}, edgeR\cite{2} or the RNA-seq workflow from Bioconductor: rnaseqGene. In brief, one usually starts with the raw FASTQ files from the RNA sequencing run. Those are then aligned to a given reference genome. At the time of writing (October 2018), we recommend the STAR aligner\cite{3}. After obtaining the aligned BAM files, one can map the reads to exons or genes of a GTF annotation file using HT-seq or by using summarizedOverlaps from GenomicAlignments. The resulting count table can then be loaded into the OUTRIDER package as we will describe below.

3 A quick tour

Here we assume that we already have a count table and no additional preprocessing needs to be done. We can start and obtain results with 3 commands. First, create an OutriderDataSet from a count table or a Summarized Experiment object. Second, run the full pipeline using the command OUTRIDER. In the third and last step the results table is extracted from the OutriderDataSet with the results function. Furthermore, analysis plots that are described in section 5 can be directly created from the OutriderDataSet object.

library(OUTRIDER)
# get data
cctsFile <- system.file('extdata', 'KremerNBaderSmall.tsv',
    package='OUTRIDER')
cctsTable <- read.table(cctsFile, check.names=FALSE)
ods <- OutriderDataSet(countData=cctsTable)

# filter out non expressed genes
ods <- filterExpression(ods, minCounts=TRUE, filterGenes=TRUE)

# run full OUTRIDER pipeline (control, fit model, calculate P-values)
ods <- OUTRIDER(ods)

## Tue Oct 24 18:23:00 2023: Iteration: 14 loss: 4.1500357971964
## Time difference of 1.306976 mins

# results (only significant)
res <- results(ods)
head(res)

## geneID sampleID pValue padjust zScore l2fc rawcounts
## 1: ATAD3C MUC1360 3.457721e-11 1.923418e-07 5.25 1.85 948
## 2: NBPF15 MUC1351 7.752762e-10 4.312611e-06 5.76 0.78 7591
## 3: MSTO1 MUC1367 4.927029e-09 2.740747e-05 -6.19 -0.81 761
## 4: HDAC1 MUC1350 8.156448e-09 4.537169e-05 -6.05 -0.80 2215
## 5: DCAF6 MUC1374 6.151571e-08 3.421920e-04 -5.71 -0.62 2348
## 6: NBPF16 MUC1351 2.415120e-07 6.717266e-04 4.83 0.68 4014

## normcounts meanCorrected theta aberrant AberrantBySample AberrantByGene
## 1: 243.49 67.27 16.87 TRUE 1 1
## 2: 7068.34 4121.80 108.94 TRUE 2 1
## 3: 729.47 1276.09 150.43 TRUE 1 1
### 4: 2090.62 3649.09 134.84 TRUE 1 1
### 5: 3077.78 4724.22 196.80 TRUE 1 1
### 6: 3845.28 2402.68 104.93 TRUE 2 1

## padj rank FDR set

## 1: 1 transcriptome-wide
## 2: 1 transcriptome-wide
## 3: 1 transcriptome-wide
## 4: 1 transcriptome-wide
## 5: 1 transcriptome-wide
## 6: 2 transcriptome-wide

# example of a Q-Q plot for the most significant outlier

plotQQ(ods, res[1, geneID])

---

4 An OUTRIDER analysis in detail

Apart from running the full pipeline using the single wrapper function OUTRIDER, the analysis can also be run step by step. The wrapper function does not include any preprocessing functions. Discarding non expressed genes or samples failing quality measurements should be done manually before running the OUTRIDER function or starting the analysis pipeline.

In this section we will explain the analysis functions step by step.

For this tutorial we will use the rare disease data set from Kremer et al.[4]. For testing purposes, this package contains a small subset of it.
4.1 OutriderDataSet

To use OUTRIDER create an OutriderDataSet, which derives from a RangedSummarizedExperiment object. The OutriderDataSet can be created by supplying a count matrix and optional sample annotation matrices. Alternatively, an existing Summarized experiment object from other Bioconductor backages can be used.

```r
# small testing data set
odsSmall <- makeExampleOutriderDataSet(dataset="Kremer")

# full data set from Kremer et al.
baseURL <- paste0("https://static-content.springer.com/esm/",
                  "art%3A10.1038%2Fncomms15824/MediaObjects/")
count_URL <- paste0(baseURL, "41467_2017_BFncomms15824_MOESM390_ESM.txt")
anno_URL <- paste0(baseURL, "41467_2017_BFncomms15824_MOESM397_ESM.txt")

ctsTable <- read.table(count_URL, sep="\t")
annoTable <- read.table(anno_URL, sep="\t", header=TRUE)
annoTable$sampleID <- annoTable$RNA_ID

# create OutriderDataSet object
ods <- OutriderDataSet(countData=ctsTable, colData=annoTable)
```

4.2 Preprocessing

It is recommended to preprocess the data before fitting. Our model requires that for every gene at least one sample has a non-zero count and that we observe at least one read for every 100 samples. Therefore, all genes that are not expressed must be discarded.

We provide the function `filterExpression` to remove genes that have low FPKM (Fragments Per Kilobase of transcript per Million mapped reads) expression values. The needed annotation to estimate FPKM values from the counts should be the same as for the counting. Here, we normalize by the total exon length of a gene. To do so the joint length of all exons needs to be provided. When providing a gtf, gff or TxDb object to the `filterExpression`, we extract this information automatically. But therefore the geneID’s of the count table and the gtf need to match.

By default the cutoff is set to an FPKM value of one and only the filtered OutriderDataSet object is returned. If required, the FPKM values can be stored in the OutriderDataSet object and the full object can be returned to visualize the distribution of reads before and after filtering.

```r
# get annotation
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(org.Hs.eg.db)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
```
OUTRIDER - OUTlier in RNA-Seq finDER

```r
map <- select(org.Hs.eg.db, keys=keys(txdb, keytype = "GENEID"),
keytype="ENTREZID", columns=c("SYMBOL"))

However, the TxDb.Hsapiens.UCSC.hg19.knownGene contains only well annotated
genes. This annotation will miss a lot of genes captured by RNA-seq. To include all
predicted annotations as well as non-coding RNAs please download the txdb object
from our homepage1 or create it yourself from the UCSC website2,3.

```r
try{
  library(RMariaDB)
  library(AnnotationDbi)
  con <- dbConnect(MariaDB(), host='genome-mysql.cse.ucsc.edu',
dbname="hg19", user='genome')
  map <- dbGetQuery(con, 'select kgId AS TXNAME, geneSymbol from kgXref')

txdbUrl <- paste0("https://cmm.in.tum.de/public/paper/mitoMultiOmics/ucsc.knownGenes.db")
download.file(txdbUrl, "ucsc.knownGenes.db")
txdb <- loadDb("ucsc.knownGenes.db")
}
```

# calculate FPKM values and label not expressed genes
ods <- filterExpression(ods, txdb, mapping=map,
  filterGenes=FALSE, savefpkm=TRUE)

# display the FPKM distribution of counts.
plotFPKM(ods)

1https://cmm.in.tum.de/public/paper/mitoMultiOmics/ucsc.knownGenes.db
2https://genome.ucsc.edu/cgi-bin/hgTables
3http://genomewiki.ucsc.edu/index.php/Genes_in_gtf_or_gff_format
# display gene filter summary statistics
plotExpressedGenes(ods)

# do the actual subsetting based on the filtering labels
ods <- ods[ncols(ods)$passedFilter,]
4.3 Controlling for Confounders

The next step in any analysis workflow is to visualize the correlations between samples. In most RNA-seq experiments correlations between the samples can be observed. These are often due to technical confounders (e.g. sequencing batch) or biological confounders (e.g. sex, age). These confounders can adversely affect the detection of aberrant features. Therefore, we provide options to control for them.

```r
# Heatmap of the sample correlation
# it can also annotate the clusters resulting from the dendrogram
ods <- plotCountCorHeatmap(ods, colGroups=c("SEX", "RNA_HOX_GROUP"),
normalized=FALSE, nRowCluster=4)
```

Count correlation heatmap

```r
# Heatmap of the gene/sample expression
ods <- plotCountGeneSampleHeatmap(ods, colGroups=c("SEX", "RNA_HOX_GROUP"),
normalized=FALSE, nRowCluster=4)
```
We have different ways to control for confounders present in the data. The first and standard way is to calculate the \texttt{sizeFactors} as done in \textit{DESeq2}[1].

Additionally, the \texttt{controlForConfounders} function calls an autoencoder that automatically controls for confounders present in the data. Therefore an encoding dimension $q$ needs to be set or the default value 20 is used. The optimal value of $q$ can be determined using the \texttt{findEncodingDim} function. After controlling for confounders, the heatmap should be plotted again. If it worked, no batches should be present and the correlations between samples should be reduced and close to zero.

```r
# automatically control for confounders
# we use only 3 iterations to make the vignette faster. The default is 15.
ods <- estimateSizeFactors(ods)
ods <- controlForConfounders(ods, q=21, iterations=3)
## Time difference of 2.183487 mins
```
Alternatively, other methods can be used to control for confounders. In addition to the `autoencoder`, we implemented a PCA based approach. The PCA implementation can be utilized by setting `implementation="pca"`. Also PEER can be used together with the OUTRIDER framework. A detailed description on how to do this can be found in section 6.1. Furthermore, any other method can be used by providing the `normalizationFactor` matrix. This matrix must be computed beforehand using the appropriate method. Its purpose is to normalize for technical effects or control for additional expression patterns.
4.4 Finding the right encoding dimension $q$

In the previous section, we fixed the encoding dimension $q = 21$. But having the right encoding dimension is crucial in finding outliers in the data. On the one hand, if $q$ is too big the autoencoder will learn the identity matrix and will overfit the data. On the other hand, if $q$ is too small the autoencoder cannot learn the necessary covariates existing in the data. Therefore, it is recommended for any new dataset to estimate the optimal encoding dimension to gain the best performance. With the function `findEncodingDim` one can find the optimal encoding dimension. To this end, we artificially introduce corrupted counts randomly into the dataset and monitor the performance calling those corrupted counts. The optimal dimension $q$ is then selected as the dimension maximizing the area under the precision-recall curve for identifying corrupted counts.

```r
# find the optimal encoding dimension q
ods <- findEncodingDim(ods)

# visualize the hyper parameter optimization
plotEncDimSearch(ods)
```

Since this function runs a full OUTRIDER fit for a range of encoding dimensions, it is quite CPU intensive, but can increase the overall performance of the autoencoder and is recommended for any data set. If $q$ is not provided by the user, it will be estimated based on the number of samples.

4.4.1 Excluding samples from the autoencoder fit

Since OUTRIDER expects that each sample within the population is independent of all others, replicates could mask effects specific to this sample. This is also true if trios are present in the data, where the parents can be seen as biological replicates. Here, we recommend to exclude the sample of interest or the replicates from the fitting. Later on, for all samples P-values are calculated.

In this rare disease data set we know that two samples (MUC1344 and MUC1365) have the same defect. To exclude one or both of them, we can use the `sampleExclusionMask` function.

```r
# set exclusion mask
sampleExclusionMask(ods) <- FALSE
sampleExclusionMask(ods[,"MUC1365"])) <- TRUE

# check which samples are excluded from the autoencoder fit
sampleExclusionMask(ods)
```

## 35834 57415 61695 61982 65937 66623 69245 69248 69456
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 70038 70041 72748 74123 74172 76619 76620 76621 76622
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 76623 76624 76625 76626 76627 76628 76629 76630 76631
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
4.5 Fitting the negative binomial model

The fit of the negative binomial model is done during the autoencoder fitting. This step is only needed if alternative methods to control the data are used. To fit the dispersion and the mean, the `fit` function is applied to the `OutriderDataSet`.

```r
# fit the model when alternative methods where used in the control step
ods <- fit(ods)
hist(theta(ods))
```

4.6 P-value calculation

After determining the fit parameters, two-sided P-values are computed using the following equation:

\[
p_{ij} = 2 \cdot \min \left\{ \frac{1}{2} \sum_{0}^{k_{ij}} NB(\mu_{ij}, \theta_i), 1 - \sum_{0}^{k_{ij}-1} NB(\mu_{ij}, \theta_i) \right\},
\]

where \( NB(\mu, \theta) \) denotes the negative binomial distribution.
where the $\frac{1}{2}$ term handles the case of both terms exceeding 0.5, which can happen due to the discrete nature of counts. Here $\mu_{ij}$ are computed as the product of the fitted correction values from the autoencoder and the fitted mean adjustments. If required a one-sided test can be performed using the argument `alternative` and specifying 'less' or 'greater' depending on the research question. Multiple testing correction is done across all genes in a per-sample fashion using Benjamini-Yekutieli’s false discovery rate method [5]. Alternatively, all adjustment methods supported by `p.adjust` can be used via the `method` argument.

```r
# compute P-values (nominal and adjusted)
ods <- computePvalues(ods, alternative="two.sided", method="BY")
```

4.7 Z-score calculation

The Z-scores on the log transformed counts can be used for visualization, filtering, and ranking of samples. By running the `computeZscores` function, the Z-scores are computed and stored in the `OutriderDataSet` object. The Z-scores are calculated using:

$$z_{ij} = \frac{l_{ij} - \mu_j^l}{\sigma_j}$$

$$l_{ij} = \log_2 \left( \frac{k_{ij} + 1}{c_{ij} + 1} \right),$$

where $\mu_j^l$ is the mean and $\sigma_j^l$ the standard deviation of gene $j$ and $l_{ij}$ is the log transformed count after correction for confounders.

```r
# compute the Z-scores
ods <- computeZscores(ods)
```

5 Results

The `OUTRIDER` package offers multiple ways to display the results. It creates a results table containing all the values computed during the analysis. Furthermore, it offers various plot functions that guide the user through the analysis.

5.1 Results table

The `results` function gathers all the previously computed values and combines them into one table.

```r
# get results (default only significant, padj < 0.05)
res <- results(ods)
head(res)
```
## geneID sampleID pValue padjust zScore l2fc rawcounts
## 1: NUDT12 65937 1.996327e-22 2.351837e-17 -10.34 -7.96 0
## 2: STAG2 MUC0490 3.489208e-21 4.110574e-16 -9.74 -1.87 622
## 3: TALDO1 MUC1427 7.141810e-18 8.413640e-13 -9.43 -3.28 482
## 4: MCOLN1 MUC1361 1.020249e-15 1.201937e-10 -8.81 -2.41 150
## 5: RETSAT MUC1374 1.183302e-15 1.394027e-10 -8.94 -3.64 76
## 6: NDUFA10 MUC1358 1.697930e-15 1.819958e-10 -8.35 -1.14 1137
## normcounts meanCorrected theta aberrant AberrantBySample AberrantByGene
## 1: 0.00 453.42 19.29 TRUE 2 1
## 2: 1108.01 4066.03 83.46 TRUE 6 1
## 3: 470.04 4556.67 27.89 TRUE 1 1
## 4: 157.82 841.03 42.05 TRUE 1 1
## 5: 148.33 1869.83 21.81 TRUE 2 1
## 6: 1056.11 2333.17 139.99 TRUE 5 1
## padj rank FDR set
## 1: 1.0 transcriptome-wide
## 2: 1.0 transcriptome-wide
## 3: 1.0 transcriptome-wide
## 4: 1.0 transcriptome-wide
## 5: 1.0 transcriptome-wide
## 6: 1.5 transcriptome-wide
dim(res)
## [1] 256 15

# setting a different significance level and filtering by Z-scores
res <- results(ods, padjCutoff=0.1, zScoreCutoff=2)
head(res)
## geneID sampleID pValue padjust zScore l2fc rawcounts
## 1: NUDT12 65937 1.996327e-22 2.351837e-17 -10.34 -7.96 0
## 2: STAG2 MUC0490 3.489208e-21 4.110574e-16 -9.74 -1.87 622
## 3: TALDO1 MUC1427 7.141810e-18 8.413640e-13 -9.43 -3.28 482
## 4: MCOLN1 MUC1361 1.020249e-15 1.201937e-10 -8.81 -2.41 150
## 5: RETSAT MUC1374 1.183302e-15 1.394027e-10 -8.94 -3.64 76
## 6: NDUFA10 MUC1358 1.697930e-15 1.819958e-10 -8.35 -1.14 1137
## normcounts meanCorrected theta aberrant AberrantBySample AberrantByGene
## 1: 0.00 453.42 19.29 TRUE 2 1
## 2: 1108.01 4066.03 83.46 TRUE 6 1
## 3: 470.04 4556.67 27.89 TRUE 1 1
## 4: 157.82 841.03 42.05 TRUE 1 1
## 5: 148.33 1869.83 21.81 TRUE 2 1
## 6: 1056.11 2333.17 139.99 TRUE 5 1
## padj rank FDR set
## 1: 1.0 transcriptome-wide
## 3: 1.0 transcriptome-wide
## 4: 1.0 transcriptome-wide
## 5: 1.0 transcriptome-wide
## 6: 1.5 transcriptome-wide

dim(res)
## [1] 315 15

### 5.2 Number of aberrant genes per sample

One quantity of interest is the number of aberrantly expressed genes per sample. This can be displayed using the plotting function `plotAberrantPerSample`. Alternatively, the function `aberrant` can be used to identify aberrant events, which can be summed by sample or gene using the parameter `by`. These numbers depend on the cutoffs, which can be specified in both functions (`padjCutoff` and `zScoreCutoff`).

```r
# number of aberrant genes per sample
tail(sort(aberrant(ods, by="sample")))
## MUC1367 MUC1381 MUC1368 MUC1363 MUC1364 76633
## 6 6 7 9 10 36
tail(sort(aberrant(ods, by="gene", zScoreCutoff=1)))
## ZFAT DNAJC3 SLMO2-ATP5E NXT2 RPS4Y1 PRKY
## 2 2 2 2 2 2
# plot the aberrant events per sample
plotAberrantPerSample(ods, padjCutoff=0.05)
```
5.3 Volcano plots

To view the distribution of P-values on a sample level, volcano plots can be displayed. Most of the plots make use of the plotly framework to create interactive plots. To only use basic R functionality from graphics the basePlot argument can be set to TRUE.

```r
# MUC1344 is a diagnosed sample from Kremer et al.
plotVolcano(ods, "MUC1344", basePlot=TRUE)
```
5.4 Gene level plots

Additionally, we include two plots at the gene level. `plotExpressionRank` plots the counts in ascending order. By default, the controlled counts are plotted. To plot raw counts, the argument `normalized` can be set to `FALSE`.

When using the `plotly` framework for plotting, all computed values are displayed for each data point. The user can access this information by hovering over each data point with the mouse.

```
# expression rank of a gene with outlier events
plotExpressionRank(ods, "TIMMDC1", basePlot=TRUE)
```
The quantile-quantile plot can be used to see whether the fit converged well. In presence of an outlier, it can happen that most of the points end up below the confidence band. This is fine and indicates that we have conservative P-values for the other points. Here is an example with two outliers:

```r
## QQ-plot for a given gene
plotQQ(ods, "TIMMDC1")
```

Since we do test how fare the observed count is away from the expected expression level, it is also helpful to visualize the predictions against the observed counts.
6 Additional features

6.1 Using PEER to control for confounders

PEER[6] is a well known tool to control for unknown effects in RNA-seq data. PEER is only available through the peer GitHub repository. The R source code can be downloaded from here: https://github.com/downloads/PMBio/peer/R_peer_source_1.3.tgz. The installation of the package has to be done manually by the user. After the installation one can use the following function to control for confounders with PEER.

```r
#' PEER implementation
#'
peer <- function(ods, maxFactors=NA, maxItr=1000){

  # check for PEER
  if(!require(peer)){
    stop("Please install the 'peer' package from GitHub to use this ",
         "functionality.")
  }

  # default and recommendation by PEER: min(0.25*n, 100)
```
if(is.na(maxFactors)){
  maxFactors <- min(as.integer(0.25* ncol(ods)), 100)
}

# log counts
logCts <- log2(t(t(counts(ods)+1)/sizeFactors(ods)))

# prepare PEER model
model <- PEER()
PEER_setNmax_iterations(model, maxItr)
PEER_setNk(model, maxFactors)
PEER_setPhenoMean(model, logCts)
PEER_setAdd_mean(model, TRUE)

# run fullpeer pipeline
PEER_update(model)

# extract PEER data
peerResiduals <- PEER_getResiduals(model)
peerMean <- t(t(2^(logCts - peerResiduals)) * sizeFactors(ods))

# save model in object
normalizationFactors(ods) <- pmax(peerMean, 1E-8)
metadata(ods)[["PEER_model"]]<- list(
  alpha = PEER_getAlpha(model),
  residuals = PEER_getResiduals(model),
  W = PEER_getW(model))

return(ods)
}

With the function above we can run the full OUTRIDER pipeline as follows:

# Control for confounders with PEER
ods <- estimateSizeFactors(ods)
ods <- peer(ods)
ods <- fit(ods)
ods <- computeZscores(ods, peerResiduals=TRUE)
ods <- computePvalues(ods)

# Heatmap of the sample correlation after controlling
ods <- plotCountCorHeatmap(ods, normalized=TRUE)
### Power analysis

We provide the `plotPowerAnalysis` function to show, what kind of changes can be significant depending on the mean count.

```r
## P-values versus Mean Count
plotPowerAnalysis(ods)
```

Here, we see that it is only for sufficiently high expressed genes possible, to obtain significant P-values, especially for the downregulation cases.

## References


**Session info**

Here is the output of `sessionInfo()` on the system on which this document was compiled:

```r
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 22.04.3 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.18-bioc/R/lib/libRblas.so
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
##
## locale:
## [1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
## [3] LC_TIME=en_GB LC_COLLATE=C
## [5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=en_US.UTF-8 LC_NAME=C
## [9] LC_ADDRESS=C LC_TELEPHONE=C
##
## time zone: America/New_York
## tzcode source: system (glibc)

## attached base packages:
```
<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>abind</td>
<td>1.4-5</td>
</tr>
<tr>
<td>compiler</td>
<td>4.3.1</td>
</tr>
<tr>
<td>withr</td>
<td>2.5.1</td>
</tr>
<tr>
<td>bit64</td>
<td>4.0.5</td>
</tr>
<tr>
<td>backports</td>
<td>1.4.1</td>
</tr>
<tr>
<td>viridis</td>
<td>0.6.4</td>
</tr>
<tr>
<td>DBI</td>
<td>1.1.3</td>
</tr>
<tr>
<td>heatmaply</td>
<td>1.5.0</td>
</tr>
<tr>
<td>dendextend</td>
<td>1.17.1</td>
</tr>
<tr>
<td>highr</td>
<td>0.10</td>
</tr>
<tr>
<td>biomaRt</td>
<td>2.58.0</td>
</tr>
<tr>
<td>rappdirs</td>
<td>0.3.3</td>
</tr>
<tr>
<td>DelayedArray</td>
<td>0.28.0</td>
</tr>
<tr>
<td>rjson</td>
<td>0.2.21</td>
</tr>
<tr>
<td>tools</td>
<td>4.3.1</td>
</tr>
<tr>
<td>glue</td>
<td>1.6.2</td>
</tr>
<tr>
<td>restfulr</td>
<td>0.0.15</td>
</tr>
<tr>
<td>nlme</td>
<td>3.1-163</td>
</tr>
<tr>
<td>grid_4.3.1</td>
<td></td>
</tr>
<tr>
<td>checkmate_2.2.0</td>
<td></td>
</tr>
<tr>
<td>reshape2_1.4.4</td>
<td></td>
</tr>
<tr>
<td>generics_0.1.3</td>
<td></td>
</tr>
<tr>
<td>gtable_0.3.4</td>
<td></td>
</tr>
<tr>
<td>ca_0.71.1</td>
<td></td>
</tr>
<tr>
<td>tidyr_1.3.0</td>
<td></td>
</tr>
<tr>
<td>hms_1.1.3</td>
<td></td>
</tr>
<tr>
<td>xml2_1.3.5</td>
<td></td>
</tr>
<tr>
<td>utf8_1.2.4</td>
<td></td>
</tr>
<tr>
<td>XVector_0.42.0</td>
<td></td>
</tr>
<tr>
<td>foreach_1.5.2</td>
<td></td>
</tr>
<tr>
<td>pillar_1.9.0</td>
<td></td>
</tr>
<tr>
<td>stringr_1.5.0</td>
<td></td>
</tr>
<tr>
<td>splines_4.3.1</td>
<td></td>
</tr>
<tr>
<td>dplyr_1.1.3</td>
<td></td>
</tr>
<tr>
<td>BBmisc_1.13</td>
<td>BiocFileCache_2.10.0</td>
</tr>
<tr>
<td>lattice_0.22.5</td>
<td></td>
</tr>
<tr>
<td>tidyselect_1.2.0</td>
<td></td>
</tr>
<tr>
<td>bit_4.0.5</td>
<td></td>
</tr>
<tr>
<td>registry_0.5.1</td>
<td></td>
</tr>
<tr>
<td>locfit_1.5-9.8</td>
<td></td>
</tr>
<tr>
<td>Biostrings_2.70.0</td>
<td></td>
</tr>
<tr>
<td>gridExtra_2.3</td>
<td></td>
</tr>
<tr>
<td>xfun_0.40</td>
<td></td>
</tr>
<tr>
<td>pheatmap_1.0.12</td>
<td></td>
</tr>
<tr>
<td>stringi_1.7.12</td>
<td></td>
</tr>
<tr>
<td>lazyeval_0.2.2</td>
<td></td>
</tr>
<tr>
<td>yaml_2.3.7</td>
<td></td>
</tr>
<tr>
<td>evaluate_0.22</td>
<td></td>
</tr>
<tr>
<td>codetools_0.2.19</td>
<td></td>
</tr>
<tr>
<td>tibble_3.2.1</td>
<td></td>
</tr>
<tr>
<td>BiocManager_1.30.22</td>
<td></td>
</tr>
<tr>
<td>cli_3.6.1</td>
<td></td>
</tr>
<tr>
<td>munsell_0.5.0</td>
<td>Rcpp_1.0.11</td>
</tr>
<tr>
<td>dplyr_2.3.4</td>
<td></td>
</tr>
<tr>
<td>png_0.1-8</td>
<td></td>
</tr>
<tr>
<td>XML_3.99-0.14</td>
<td></td>
</tr>
<tr>
<td>parallel_4.3.1</td>
<td></td>
</tr>
<tr>
<td>ggplot2_3.4.4</td>
<td></td>
</tr>
<tr>
<td>assertthat_0.2.1</td>
<td></td>
</tr>
<tr>
<td>blob_1.2.4</td>
<td>prettyunits_1.2.0</td>
</tr>
<tr>
<td>bitops_1.0-7</td>
<td>viridisLite_0.4.2</td>
</tr>
<tr>
<td>scales_1.2.1</td>
<td>purrrr_1.0.2</td>
</tr>
<tr>
<td>crayon_1.5.2</td>
<td>BiocStyle_2.30.0</td>
</tr>
<tr>
<td>rlang_1.1.1</td>
<td>KEGGREST_1.42.0</td>
</tr>
<tr>
<td>TSP_1.2-4</td>
<td></td>
</tr>
</tbody>
</table>