`edge`:
<u>E</u>xtraction of <u>D</u>ifferential <u>G</u>ene Expression
Version 2.43.0

John D. Storey[*1,2], Jeffrey T. Leek[3], and Andrew J. Bass[1]

[1]Lewis-Sigler Institute for Integrative Genomics, Princeton University, Princeton NJ USA
[2]Center for Statistics and Machine Learning, Princeton University, Princeton NJ USA
[3]Department of Biostatistics, John Hopkins University, Baltimore MD USA

February 21, 2026

First edition: October 2005

Most recent edition: March 2015

Note: `edge` was first released in 2005 and described in the publication Leek et al. (2006). It was an independently released R package by the John Storey Lab, which included multi-threading and a graphical user interface. However, the current version is now available through Bioconductor as a standard R package.

---

[*]http://genomine.org/contact.html

Contents

1 Introduction

The `edge` package implements methods for carrying out differential expression analyses of genome-wide gene expression studies. Significance testing using the optimal discovery procedure and generalized likelihood ratio test (equivalent to F-tests and t-tests) are implemented for general study designs. Special functions are available to facilitate the analysis of common study designs, including time course experiments. Other

packages such as `sva` and `qvalue` are integrated in `edge` to provide a wide range of tools for gene expression analysis.

`edge` performs significance analysis by using a framework developed by Storey (2007) called the optimal discovery procedure (ODP). Whereas standard methods employ statistics that are essentially designed for testing one gene at a time (e.g., t-statistics and F-statistics), the ODP-statistic uses information across all genes to test for differential expression. Storey et al. (2007) shows that the ODP is a principled, often times more powerful, approach to multiple hypothesis testing compared to traditional methods. The improvements in power from using the optimal discovery procedure can be substantial; Figure 1 shows a comparison between `edge` and five leading software packages based on the Hedenfalk et al. (2001) breast cancer expression study.
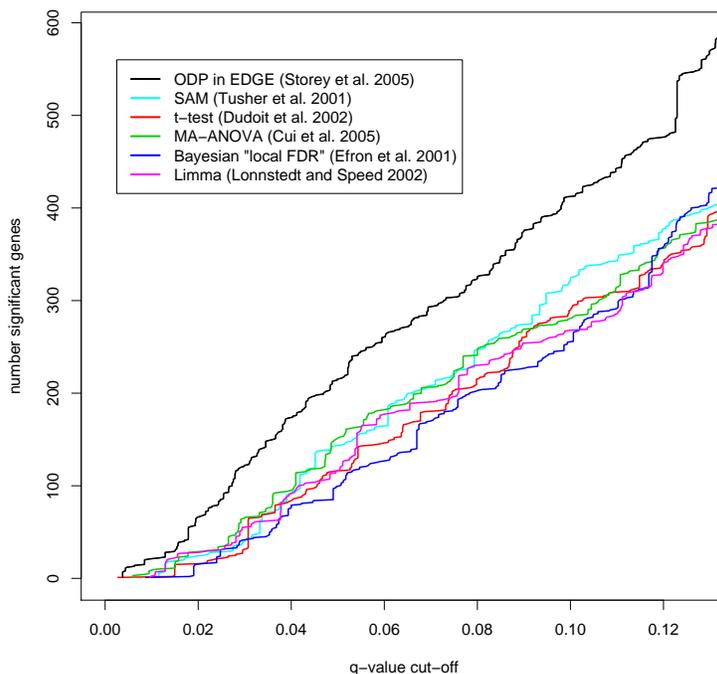
Figure 1: Comparison of EDGE to various other leading methods for identifying differential expressed genes in the Hedenfalk et al. (2001) study. This figure is from Leek et al. (2006).

`edge` also implements strategies that have been specifically designed for time course experiments and other complex study designs. Specifically, Storey et al. (2005) developed a procedure that simplifies the modelling process for time course experiments. In addition to identifying differentially expressed genes in multi-class, time course, and general study designs, `edge` includes implementations of popular packages such as `sva` and `qvalue` to help simplify the analysis process for researchers.

The rest of the document details how to use `edge` in three different case studies: static sampling among K groups, independent time course, and longitudinal time course. For additional information regarding the optimal discovery procedure or the Storey et al. (2005) methodology for time course experiments, see section 2.

2 Citing this package

When reporting results involving the estimation of false discovery rate or q-value quantities, please cite:

JD Storey. A direct approach to false discovery rates. *Journal of the Royal Statistical Society Series B-Statistical Methodology*, 64:479–498, 2002

John D Storey and Robert Tibshirani. Statistical significance for genomewide studies. *Proc Natl Acad Sci U S A*, 100(16):9440–5, Aug 2003. doi: 10.1073/pnas.1530509100

When reporting results involving the analysis of time course studies, please cite:

John D. Storey, Wenzhong Xiao, Jeffrey T. Leek, Ronald G. Tompkins, and Ronald W. Davis. Significance analysis of time course microarray experiments. *Proceedings of the National Academy of Sciences of the United States of America*, 102(36):12837–12842, 2005. doi: 10.1073/pnas.0504609102. URL http://www.pnas.org/content/102/36/12837.abstract

When reporting results involving use of the optimal discovery procedure (`odp`), please cite:

John D. Storey. The optimal discovery procedure: a new approach to simultaneous significance testing. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(3):347–368, 2007. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2007.005592.x. URL http://dx.doi.org/10.1111/j.1467-9868.2007.005592.x

John D. Storey, James Y. Dai, and Jeffrey T. Leek. The optimal discovery procedure for large-scale significance testing, with applications to comparative microarray experiments. *Biostatistics*, 8(2):414–432, 2007. doi: 10.1093/biostatistics/kxl019. URL http://biostatistics.oxfordjournals.org/content/8/2/414.abstract

Sangsoon Woo, Jeffrey T. Leek, and John D. Storey. A computationally efficient modular optimal discovery procedure. *Bioinformatics*, 27(4):509–515, 2011. doi: 10.1093/bioinformatics/btq701. URL http://bioinformatics.oxfordjournals.org/content/27/4/509.abstract

Andrew J Bass and John D Storey. The optimal discovery procedure for significance analysis of general gene expression studies. *Bioinformatics*, 37(3):367–374, 08 2020. doi: 10.1093/bioinformatics/btaa707. URL https://doi.org/10.1093/bioinformatics/btaa707

When reporting results involving surrogate variable analysis (`apply_sva`), please cite:

Jeffrey T Leek and John D Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genet*, 3(9):e161, 09 2007. doi: 10.1371/journal.pgen.0030161

Jeffrey T Leek and John D Storey. A general framework for multiple testing dependence. *Proc Natl Acad Sci U S A*, 105(48):18718–23, Dec 2008. doi: 10.1073/pnas.0808709105

To cite the `edge` R package itself, please type the following to retrieve the citation:

```
citation("edge")

## To cite package 'edge' in publications
## use:
##
##    Storey JD, Leek JT, Bass AJ (2025).
```

```
##    _edge: Extraction of Differential Gene
##    Expression_. R package version 2.43.0,
##    <https://github.com/jdstorey/edge>.
##
## A BibTeX entry for LaTeX users is
##
##    @Manual{,
##      title = {edge: Extraction of Differential Gene Expression},
##      author = {John D. Storey and Jeffrey T. Leek and Andrew J. Bass},
##      year = {2025},
##      note = {R package version 2.43.0},
##      url = {https://github.com/jdstorey/edge},
##    }
##
## ATTENTION: This citation information has
## been auto-generated from the package
## DESCRIPTION file and may need manual
## editing, see 'help("citation")'.
```

3   Getting help

Many questions about `qvalue` will hopefully be answered by this documentation and references therein. As with any R package, detailed information on functions, their arguments and values, can be obtained in the help files. To view the help for `qvalue` within R, type

```
help(package = "edge")
```

If you identify bugs related to basic usage please contact the authors directly, preferably via GitHub at https://github.com/jdstorey/edge/issues. Otherwise, any questions or problems regarding `edge` will most efficiently be addressed on the Bioconductor support site, https://support.bioconductor.org/.

4   Quick start guide

To get started, first load the `kidney` dataset included in the package:

```
library(edge)
data(kidney)
age <- kidney$age
sex <- kidney$sex
kidexpr <- kidney$kidexpr
```

The `kidney` study is interested in determining differentially expressed genes with respect to age in the kidney. The `age` variable is the age of the subjects and the `sex` variable is whether the subjects are male or female. The expression values for the genes are contained in the `kidexpr` variable.

Once the data has been loaded, the user has two options to create the experimental models: `build_models` or `build_study`. If the experiment models are unknown to the user, `build_study` can be used to create the models:

```
de_obj <- build_study(data = kidexpr, adj.var = sex,
    tme = age, sampling = "timecourse")
```

```
full_model <- fullModel(de_obj)
null_model <- nullModel(de_obj)
```

sampling describes the type of experiment performed, `adj.var` is the adjustment variable and `time` is the time variable in the study. If the experiment is more complex then type **?build_study** for additional arguments.

If the full and null models are known to the user then **build_models** can be used to make an **deSet** object:

```
library(splines)
cov <- data.frame(sex = sex, age = age)
null_model <- ~sex
full_model <- ~sex + ns(age, df = 4)
de_obj <- build_models(data = kidexpr, cov = cov,
    null.model = null_model, full.model = full_model)
```

`cov` is a data frame of covariates, `null.model` is the null model and `full.model` is the full model. The input `cov` is a data frame with the column names the same as the variables in the full and null models.

The `odp` or `lrt` function can be used on `de_obj` to implement either the optimal discovery procedure or the likelihood ratio test, respectively:

```
# optimal discovery procedure
de_odp <- odp(de_obj, bs.its = 50, verbose = FALSE)
# likelihood ratio test
de_lrt <- lrt(de_obj)
```

To access the π_0 estimate, p-values, q-values and local false discovery rates for each gene, use the function `qvalueObj`:

```
qval_obj <- qvalueObj(de_odp)
qvals <- qval_obj$qvalues
pvals <- qval_obj$pvalues
lfdr <- qval_obj$lfdr
pi0 <- qval_obj$pi0
```

The following sections of the manual go through various case studies for a more comprehensive overview of the `edge` package.

5 Case study: static experiment

In the static sampling experiment, the arrays have been collected from distinct biological groups without respect to time. The goal is to identify genes that have a statistically significant difference in average expression across these distinct biological groups. The example data set that will be used in this section is the `gibson` data set and it is a random subset of the data from Idaghdour et al..

The `gibson` data set provides gene expression measurements in peripheral blood leukocyte samples from three Moroccan Amazigh groups leading distinct ways of life: desert nomadic (DESERT), mountain agrarian (VILLAGE), and coastal urban (AGADIR). We are interested in finding the genes that differentiate the Moroccan Amazigh groups the most. See Idaghdour et al. for additional information regarding the data.
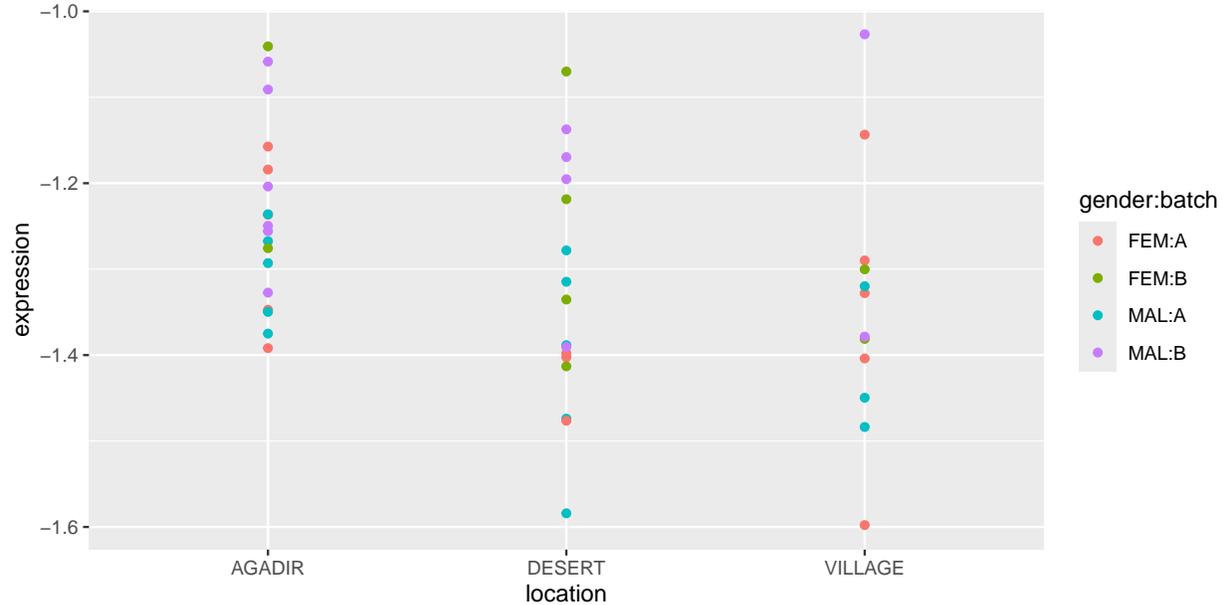
Figure 2: Plot of gene 1 in the `gibson` study.

5.1 Importing the data

To import the `gibson` data use the `data` function:

```
data(gibson)
gibexpr <- gibson$gibexpr
batch <- gibson$batch
gender <- gibson$gender
location <- gibson$location
```

There are a few variables in the data set: `batch`, `gibexpr`, `gender`, and `location`. The three covariates of interest are `gender`, `batch` and `location`. The biological variable is the `location` variable, which contains information on where individuals are sampled: "VILLAGE", "DESERT" or "AGADIR". The `gender` variable specifies whether the individual is a male or a female and there are two different `batches` in the study. The `gibexpr` variable contains the gene expression measurements.

As an example, the expression values of the first gene are shown in Figure 2. In the figure, it appears that the individuals from "VILLAGE" are more expressed when compared to the other lifestyles. We should stop short of that observation because the data needs to be adjusted with the experimental models. Before that, the full and null models of the study needs to be carefully formulated.

5.2 Creating the full and null models

In order to find deferentially expressed genes, there first needs to be a full and null model for the study. There are two ways to input the experimental models in `edge`: `build_models` and `build_study`. `build_study` should be used by users unfamiliar with formulating the full and null models but are familiar with the covariates in the study:

```
de_obj <- build_study(data = gibexpr, adj.var = cbind(gender,
    batch), grp = location, sampling = "static")
```

`adj.var` is for the adjustment variables, `grp` is the variable containing the group assignments for each individual in the study and `sampling` describes the type of experiment. Since `gibson` is a static study, the `sampling` argument will be "static". The `grp` variable will be the `location` variable and the adjustment variables are `gender` and `batch`.

Alternatively, if the user is familiar with their full and null models in the study then `build_models` can be used to input the models directly:

```
cov <- data.frame(Gender = gender, Batch = batch,
    Location = location)
null_model <- ~Gender + Batch
null_model <- ~Gender + Batch + Location
de_obj <- build_models(data = gibexpr, cov = cov,
    full.model = null_model, null.model = null_model)
```

The `cov` argument is a data frame of all the relevant covariates, `full.model` and `null.model` are the full and null models of the experiment, respectively. Notice that the models must be formatted as a formula and contain the same variable names as in the `cov` data frame. The null model contains the `gender` and `batch` covariates and the full model includes the `location` variable. Therefore, we are interested in testing whether the full model improves the model fit of a gene when compared to the null model. If it does not, then we can conclude that there is no significant difference between Moroccan Amazigh groups for this particular gene.

The variable `de_obj` is an `deSet` object that stores all the relevant experimental data. The `deSet` object is discussed further in the next section.

5.3   The `deSet` object

Once either `build_models` or `build_study` is used, an `deSet` object is created. To view the slots contained in the object:

```
slotNames(de_obj)

##   [1] "null.model"        "full.model"
##   [3] "null.matrix"       "full.matrix"
##   [5] "individual"        "qvalueObj"
##   [7] "experimentData"    "assayData"
##   [9] "phenoData"         "featureData"
## [11] "annotation"        "protocolData"
## [13] ".__classVersion__"
```

A description of each slot is listed below:

- `full.model`: the full model of the experiment. Contains the biological variables of interest and the adjustment variables.

- `null.model`: the null model of the experiment. Contains the adjustment variables in the experiment.

- `full.matrix`: the full model in matrix form.

- `null.matrix`: the null model in matrix form.

- `individual`: variable that keeps track of individuals (if same individuals are sampled multiple times).

- qvalueObj: qvalue list. Contains p-values, q-values and local false discovery rates of the significance analysis. See the qvalue package for more details.

- ExpressionSet: inherits the slots from ExpressionSet object.

ExpressionSet contains the expression measurements and other information from the experiment. The deSet object inherits all the functions from an ExpressionSet object. As an example, to access the expression values, one can use the function exprs or to access the covariates, pData:

```
gibexpr <- exprs(de_obj)
cov <- pData(de_obj)
```

The ExpressionSet class is a widely used object in Bioconductor and more information can be found here. See the section 10 on ExpressionSet to get a better understanding of how it integrates into the edge framework.

As an example of how to access the slots of de_obj suppose we are interested in viewing the full and null models. The models can be accessed by:

```
fullModel(de_obj)

## ~Gender + Batch + Location

nullModel(de_obj)

## ~Gender + Batch + Location
```

Next, we can extract the models in matrix form for computational analysis:

```
full_matrix <- fullMatrix(de_obj)
null_matrix <- nullMatrix(de_obj)
```

See ?deSet for additional functions to access different slots of the deSet object.

5.4 Fitting the data

The fit_models function is an implementation of least squares using the full and null models:

```
ef_obj <- fit_models(de_obj, stat.type = "lrt")
```

The stat.type argument specifies whether you want the odp or lrt fitted values. The difference between choosing "odp" and "lrt" is that "odp" centers the data by the null model fit which is necessary for downstream analysis in the optimal discovery procedure. fit_models creates another object with the following slots:

- fit.full: fitted values from the full model.

- fit.null: fitted values from null model.

- res.full: residuals from the full model.

- res.null: residuals from the null model.

- dH.full: diagonal elements in the projection matrix for the full model.

Figure 3: Plot of gene 1 in the `gibson` study after applying the full and null model fit. The "raw" column are the expression values of the original data.

- `beta.coef`: the coefficients for the full model.

- `stat.type`: statistic type used, either "odp" or "lrt".

To access the fitted coefficients of the full model in `ef_obj`:

```
betaCoef(ef_obj)
```

To access the full and null residuals:

```
alt_res <- resFull(ef_obj)
null_res <- resNull(ef_obj)
```

To access the fitted values:

```
alt_fitted <- fitFull(ef_obj)
null_fitted <- fitNull(ef_obj)
```

See `?deFit` for more details on accessing the slots in an `deFit` object. The fitted values of the first gene are shown in Figure 3. The null model fit is the average expression value across the interaction of `batch` and `sex`. The full model fit seems to pick up some differences relative to the null model. Next, we have to test whether the observed differences between the model fits are significant.

5.5   Significance analysis

Interpreting the models in a hypothesis test is very intuitive: Does the full model better fit the data when compared to the null model? For the fitted values of the first gene plotted in Figure 3, it seems that the full model fits the data better than the null model. In order to conclude that it is significant, we need to calculate the p-value. The user can use either the optimal discovery procedure or likelihood ratio test.

5.5.1 Likelihood ratio test

The `lrt` function performs a likelihood ratio test to determine p-values:

```
de_lrt <- lrt(de_obj, nullDistn = "normal")
```

If the null distribution, `nullDistn`, is calculated using "bootstrap" then residuals from the full model are re-sampled and added to the null model to simulate a distribution where there is no differential expression. Otherwise, the default input is "normal" and the assumption is that the null statistics follow a F-distribution. See `?lrt` for additional arguments.

5.5.2 Optimal discovery procedure

`odp` performs the optimal discovery procedure, which is a new approach developed by Storey et al. (2007) for optimally performing many hypothesis tests in a high-dimensional study. When testing a feature, information from all the features is utilized when testing for significance of a feature. It guarantees to maximize the number of expected true positive results for each fixed number of expected false positive results which is related to the false discovery rate.

The optimal discovery procedure can be implemented on `de_obj` by the `odp` function:

```
de_odp <- odp(de_obj, bs.its = 50, verbose = FALSE,
    n.mods = 50)
```

The number of bootstrap iterations is controlled by `bs.its`, `verbose` prints each bootstrap iteration number and `n.mods` is the number of clusters in the k-means algorithm.

A k-means algorithm is used to assign genes to groups in order to speed up the computational time of the algorithm. If `n.mods` is equal to the number of genes then the original optimal discovery procedure is used. Depending on the number of genes, this setting can take a very long time. Therefore, it is recommended to use a small `n.mods` value to substantially decrease the computational time. In Woo et al. (2011), it is shown that assigning `n.mods` to about 50 will cause a negligible loss in power. Type `?odp` for more details on the algorithm.

5.6 Significance results

The `summary` function can be used on an `deSet` object to give an overview of the analysis:

```
summary(de_odp)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 46 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 46 (46 total)
##   varLabels: gender batch grp
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
```

p-value density histogram
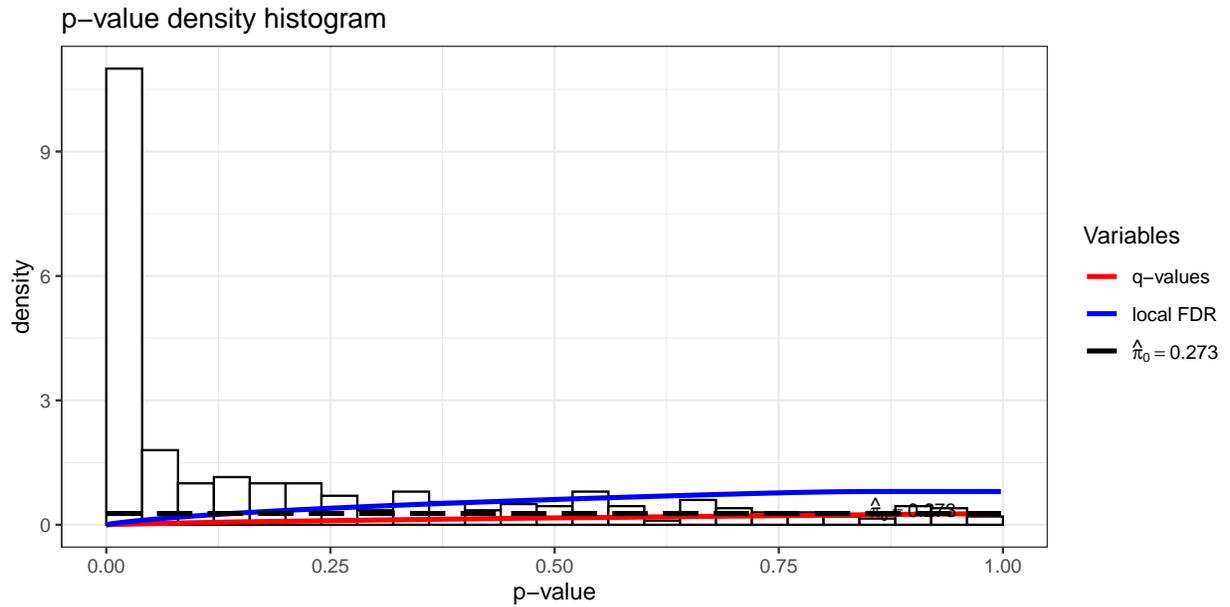
Figure 4: Applying the function `hist` to the slot `qvalueObj` in the `gibson` data set. Function is derived from the `qvalue` package.

```
##
## de Analysis Summary
##
## Total number of arrays: 46
## Total number of probes: 500
##
## Biological variables:
##  Null Model:~gender + batch
## <environment: 0x55fd91983fb0>
##
##  Full Model:~gender + batch + grp
## <environment: 0x55fd91983fb0>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.2730848
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value       50     83   147    191   231  269
## q-value        0     80   152    221   269  342
## local fdr      0     50   103    132   169  215
##             <1
## p-value    500
```

```
## q-value   500
## local fdr 500
```

There are three core summaries: `ExpressionSet` summary, `edge` analysis and statistical significance summary. The `ExpressionSet` summary shows a summary of the `ExpressionSet` object. `edge` analysis shows an overview of the models used and other information about the data set. The significance analysis shows the proportion of null genes, π_0, and significant genes at various cutoffs in terms of p-values, q-values and local false discovery rates.

The function `qvalueObj` can be used on `de_odp` to extract the significance results:

```
sig_results <- qvalueObj(de_odp)
```

The object `sig_results` is a list with the following slots:

```
names(sig_results)

##  [1] "call"       "pi0"        "qvalues"
##  [4] "pvalues"    "lfdr"       "pi0.lambda"
##  [7] "lambda"     "pi0.smooth" "stat0"
## [10] "stat"
```

The key variables are `pi0`, `pvalues`, `lfdr` and `qvalues`. The `pi0` variable provides an estimate of the proportion of null p-values, `pvalues` are the p-values, `qvalues` are the estimated q-values and `lfdr` are the local false discovery rates. Using the function `hist` on `sig_results` will produce a p-value histogram along with the density curves of q-values and local false discovery rate values:

```
hist(sig_results)
```

The plot is shown in Figure 4. To extract the p-values, q-values, local false discovery rates and the π_0 estimate:

```
pvalues <- sig_results$pvalues
qvalues <- sig_results$qvalues
lfdr <- sig_results$lfdr
pi0 <- sig_results$pi0
```

Making significance decisions based on p-values in multiple hypothesis testing problems can lead to accepting a lot of false positives in the study. Instead, using q-values to determine significant genes is recommended because it controls the false discovery rate. Q-values measure the proportion of false positives incurred when calling a particular test significant. For example, to complete our analysis of gene 1 in this example, lets view the q-value estimate:

```
qvalues[1]

## [1] 0.01716123
```

So for this particular gene, the q-value is 0.0171612. If we consider a false discovery rate cutoff of 0.1 then this gene is significant. Therefore, the observed differences observed in Figure 3 are significant so this particular gene is differentially expressed between locations.

To get a list of all the significant genes at a false discovery rate cutoff of 0.01:

```
fdr.level <- 0.01
sigGenes <- qvalues < fdr.level
```

View the qvalue vignette to get a more thorough discussion in how to use p-values, q-values, π_0 estimate and local false discovery rates to determine significant genes.

6 Case study: independent time course experiment

In the independent time course study, the arrays have been sampled with respect to time from one biological group and the goal is to identify genes that show "within-class temporal differential expression", i.e., genes that show statistically significant changes in expression over time. The example data set used in this section is a kidney data set by Rodwell et al. (2004). Gene expression measurements, from cortex and medulla samples in the kidney, were obtained from 72 human subjects ranging in age from 27 to 92 years. Only one array was obtained per sample and the age and tissue type of each subject was recorded. See Rodwell et al. (2004) for additional information regarding the data set.

6.1 Importing the data

To import the kidney data use the data function:

```
data(kidney)
age <- kidney$age
sex <- kidney$sex
kidexpr <- kidney$kidexpr
```

There are a few covariates in this data set: sex, age, and kidexpr. The two main covariates of interest are the sex and age covariates. The sex variable is whether the subject was male or female and the age variable is the age of the patients. kidexpr contains the gene expression values for the study.

As an example of a gene in the study, the expression values of the fifth gene are shown in Figure 5. It is very difficult to find a trend for this particular gene. Instead, we need to adjust the data with the models in the study which is discussed in the next section.

6.2 Creating the full and null models

In order to find differentially expressed genes, the full and null model for the study need to be formulated. There are two ways to input the experimental models in edge: build_models and build_study. build_study should be used by users unfamiliar with formulating the full and null models but are familiar with the covariates in the study:

```
de_obj <- build_study(data = kidexpr, adj.var = sex,
    tme = age, sampling = "timecourse", basis.df = 4)
```

adj.var is for the adjustment variables, tme is the time variable, basis.df is the degrees of freedom for the spline fit, and sampling describes the type of experiment. Since kidney is a time course study, the sampling argument will be "timecourse". The tme variable will be the age variable, basis.df will be 4 based on previous work by Storey et al. (2005) and the adjustment variable is sex. To view the models generated by build_study:

```
fullModel(de_obj)

## ~adj.var + ns(tme, df = 4, intercept = FALSE)
```
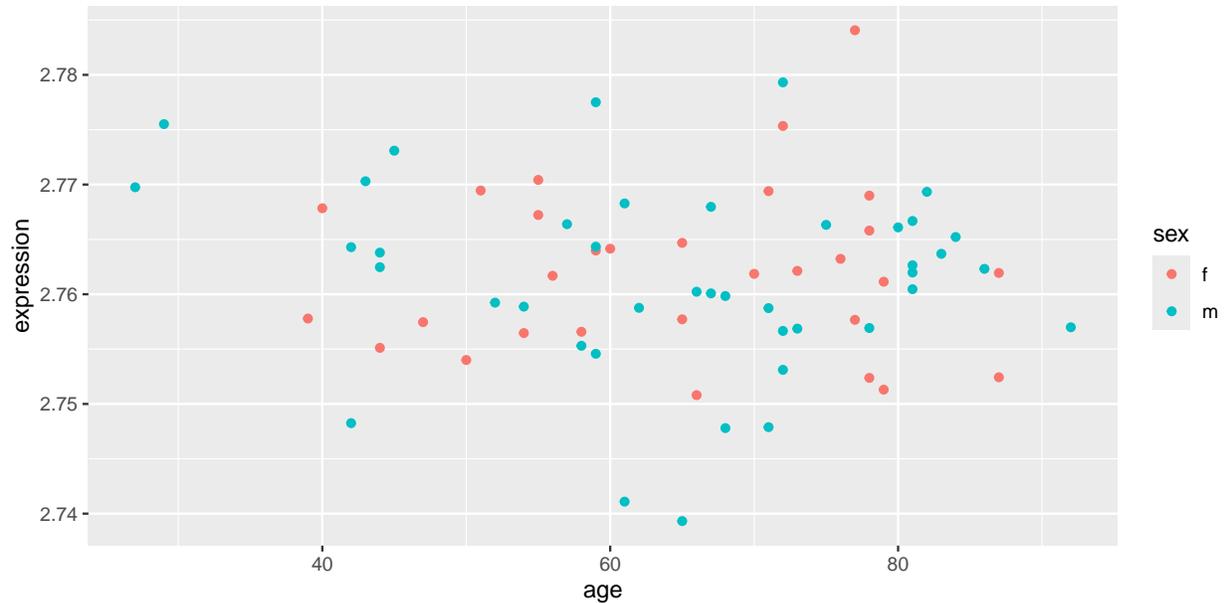
Figure 5: Plot of gene 5 in the `kidney` study.

```
## <environment: 0x55fd9d1dad60>

nullModel(de_obj)

## ~adj.var
## <environment: 0x55fd9d1dad60>
```

Notice that the difference between the full and null model is the natural spline fit of the `age` variable. If we look at Figure 5, it becomes evident that a spline curve can be used to approximate the fit of the data, and 4 degrees of freedom is chosen based on previous analysis of the expression patterns. See Storey et al. (2005) for a detailed discussion on modelling in time course studies.

Alternatively, if the user is familiar with their full and null models in the study then `build_models` can be used to input the models directly:

```
library(splines)
cov <- data.frame(sex = sex, age = age)
null_model <- ~sex
null_model <- ~sex + ns(age, df = 4)
de_obj <- build_models(data = kidexpr, cov = cov,
    full.model = null_model, null.model = null_model)
```

The `cov` argument is a data frame of all the relevant covariates, `full.model` and `null.model` are the full and null models of the experiment, respectively. Notice that the models must be formatted as a formula and contain the same variable names as in the `cov` data frame. The null model contains the `sex` covariate and the full model includes the `age` variable. Therefore, we are interested in testing whether the full model improves the model fit of a gene significantly when compared to the null model. If it does not, then we can conclude that there is no significant difference in the gene as it ages in the cortex.

15

The variable `de_obj` is an `deSet` object that stores all the relevant experimental data. The `deSet` object is discussed further in the next section.

6.3 The `deSet` object

Once either `build_models` or `build_study` is used, an `deSet` object is created. To view the slots contained in the object:

```
slotNames(de_obj)

##  [1] "null.model"        "full.model"
##  [3] "null.matrix"       "full.matrix"
##  [5] "individual"        "qvalueObj"
##  [7] "experimentData"    "assayData"
##  [9] "phenoData"         "featureData"
## [11] "annotation"        "protocolData"
## [13] ".__classVersion__"
```

A description of each slot is listed below:

- `full.model`: the full model of the experiment. Contains the biological variables of interest and the adjustment variables.

- `null.model`: the null model of the experiment. Contains the adjustment variables in the experiment.

- `full.matrix`: the full model in matrix form.

- `null.matrix`: the null model in matrix form.

- `individual`: variable that keeps track of individuals (if same individuals are sampled multiple times).

- `qvalueObj`: `qvalue` list. Contains p-values, q-values and local false discovery rates of the significance analysis. See the qvalue package for more details.

- `ExpressionSet`: inherits the slots from `ExpressionSet` object.

`ExpressionSet` contains the expression measurements and other information from the experiment. The `deSet` object inherits all the functions from an `ExpressionSet` object. As an example, to access the expression values, one can use the function `exprs` or to access the covariates, `pData`:

```
gibexpr <- exprs(de_obj)
cov <- pData(de_obj)
```

The `ExpressionSet` class is a widely used object in Bioconductor and more information can be found here. See the section 10 on `ExpressionSet` to get a better understanding of how it integrates into the `edge` framework.

As an example of how to access the slots of `de_obj` suppose we are interested in viewing the full and null models. The models can be accessed by:

```
fullModel(de_obj)
nullModel(de_obj)
```

Next, we can extract the models in matrix form for computational analysis:
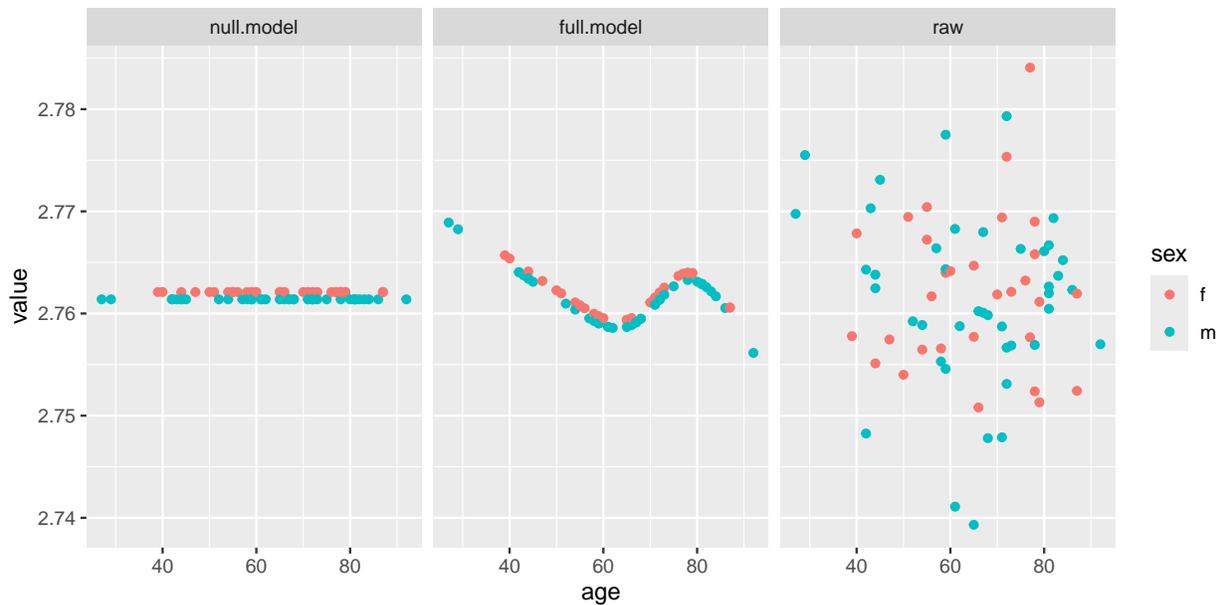
Figure 6: Plot of gene 5 in the `kidney` study after applying the full and null model fit. The "raw" column are the expression values of the original data.

```
full_matrix <- fullMatrix(de_obj)
null_matrix <- nullMatrix(de_obj)
```

See `?deSet` for additional functions to access different slots of the `deSet` object.

6.4 Fitting the data

The `fit_models` function is an implementation of least squares using the full and null models:

```
ef_obj <- fit_models(de_obj, stat.type = "lrt")
```

The `stat.type` argument specifies whether you want the `odp` or `lrt` fitted values. The difference between choosing "odp" and "lrt" is that "odp" centers the data by the null model fit which is necessary for downstream analysis in the optimal discovery procedure. `fit_models` creates another object with the following slots:

- `fit.full`: fitted values from the full model.

- `fit.null`: fitted values from null model.

- `res.full`: residuals from the full model.

- `res.null`: residuals from the null model.

- `dH.full`: diagonal elements in the projection matrix for the full model.

- `beta.coef`: the coefficients for the full model.

17

- **stat.type**: statistic type used, either "odp" or "lrt".

To access the fitted coefficients of the full model in **ef_obj**:

```
betaCoef(ef_obj)
```

To access the full and null residuals:

```
alt_res <- resFull(ef_obj)
null_res <- resNull(ef_obj)
```

To access the fitted values:

```
alt_fitted <- fitFull(ef_obj)
null_fitted <- fitNull(ef_obj)
```

See **?deFit** for more details on accessing the slots in a **deFit** object. The fitted values of the fifth gene are shown in Figure 6. The null model fit is the average expression. It appears that the full model fits the raw data better than the null model. Next, we have to test whether the observed differences between the model fits are significant.

6.5 Significance analysis

Interpreting the models in a hypothesis test is very intuitive: Does the full model better fit the data when compared to the null model? For the fitted values of the fifth gene plotted in Figure 6, it seems that the full model fits the data better than the null model. In order to conclude it is significant, we need to calculate the p-value. The user can use either the optimal discovery procedure or likelihood ratio test.

6.5.1 Likelihood ratio test

The **lrt** function performs a likelihood ratio test to determine p-values:

```
de_lrt <- lrt(de_obj, nullDistn = "normal")
```

If the null distribution, **nullDistn**, is calculated using "bootstrap" then residuals from the full model are re-sampled and added to the null model to simulate a distribution where there is no differential expression. Otherwise, the default input is "normal" and the assumption is that the null statistics follow a F-distribution. See **?lrt** for additional arguments.

6.5.2 Optimal discovery procedure

odp performs the optimal discovery procedure, which is a new approach developed by Storey et al. (2005) for optimally performing many hypothesis tests in a high-dimensional study. When testing a feature, information from all the features is utilized when testing for significance of a feature. It guarantees to maximize the number of expected true positive results for each fixed number of expected false positive results which is related to the false discovery rate.

The optimal discovery procedure can be implemented on **de_obj** by the **odp** function:

```
de_odp <- odp(de_obj, bs.its = 50, verbose = FALSE,
    n.mods = 50)
```

The number of bootstrap iterations is controlled by **bs.its**, **verbose** prints each bootstrap iteration number and **n.mods** is the number of clusters in the k-means algorithm.

A k-means algorithm is used to assign genes to groups in order to speed up the computational time of the algorithm. If `n.mods` is equal to the number of genes then the original optimal discovery procedure is used. Depending on the number of genes, this setting can take a very long time. Therefore, it is recommended to use a small `n.mods` value to substantially decrease the computational time. In Woo et al. (2011), it is shown that assigning `n.mods` to about 50 will cause a negligible loss in power. Type `?odp` for more details on the algorithm.

6.6 Significance results

The `summary` function can be used on an `deSet` object to give an overview of the analysis:

```
summary(de_odp)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 72 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: adj.var tme
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## de Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 500
##
## Biological variables:
##  Null Model:~adj.var
## <environment: 0x55fda0845578>
##
##  Full Model:~adj.var + ns(tme, df = 4, intercept = FALSE)
## <environment: 0x55fda0845578>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.4373442
##
## Cumulative number of significant calls:
##
##          <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value       4      6    29     56    95  142
## q-value       0      0     6      6    22   72
## local fdr     0      0     3      6    10   30
```

```
##             <1
## p-value    500
## q-value    500
## local fdr 500
```

There are three core summaries: `ExpressionSet` summary, `edge` analysis and statistical significance summary. The `ExpressionSet` summary shows a summary of the `ExpressionSet` object. `edge` analysis shows an overview of the models used and other information about the data set. The significance analysis shows the proportion of null genes, π_0, and significant genes at various cutoffs in terms of p-values, q-values and local false discovery rates.

The function `qvalueObj` can be used on `de_odp` to extract the significance results:

```
sig_results <- qvalueObj(de_odp)
```

The object `sig_results` is a list with the following slots:

```
names(sig_results)

##  [1] "call"       "pi0"        "qvalues"
##  [4] "pvalues"    "lfdr"       "pi0.lambda"
##  [7] "lambda"     "pi0.smooth" "stat0"
## [10] "stat"
```

The key variables are `pi0`, `pvalues`, `lfdr` and `qvalues`. The `pi0` variable provides an estimate of the proportion of null p-values, `pvalues` are the p-values, `qvalues` are the estimated q-values and `lfdr` are the local false discovery rates. Using the function `hist` on `sig_results` will produce a p-value histogram along with the density curves of q-values and local false discovery rate values:

```
hist(sig_results)
```

The plot is shown in Figure 7. To extract the p-values, q-values, local false discovery rates and the π_0 estimate:

```
pvalues <- sig_results$pvalues
qvalues <- sig_results$qvalues
lfdr <- sig_results$lfdr
pi0 <- sig_results$pi0
```

Making significance decisions based on p-values in multiple hypothesis testings problems can lead to accepting a lot of false positives in the study. Instead, using q-values to determine significant genes is recommended because it controls the false discovery rate at a level `alpha`. Q-values measure the proportion of false positives incurred when calling a particular test significant. For example, to complete our analysis of gene 5 in this example, lets view the q-value estimate:

```
qvalues[5]

## [1] 0.1789628
```

So for this particular gene, the q-value is 0.1789628. If we consider a false discovery rate cutoff of 0.1 then this gene is not significant. Therefore, the observed differences observed in Figure 6 are not significant so this particular gene is not differentially expressed as the kidney ages.

To get a list of all the significant genes at a false discovery rate cutoff of 0.1:

p−value density histogram
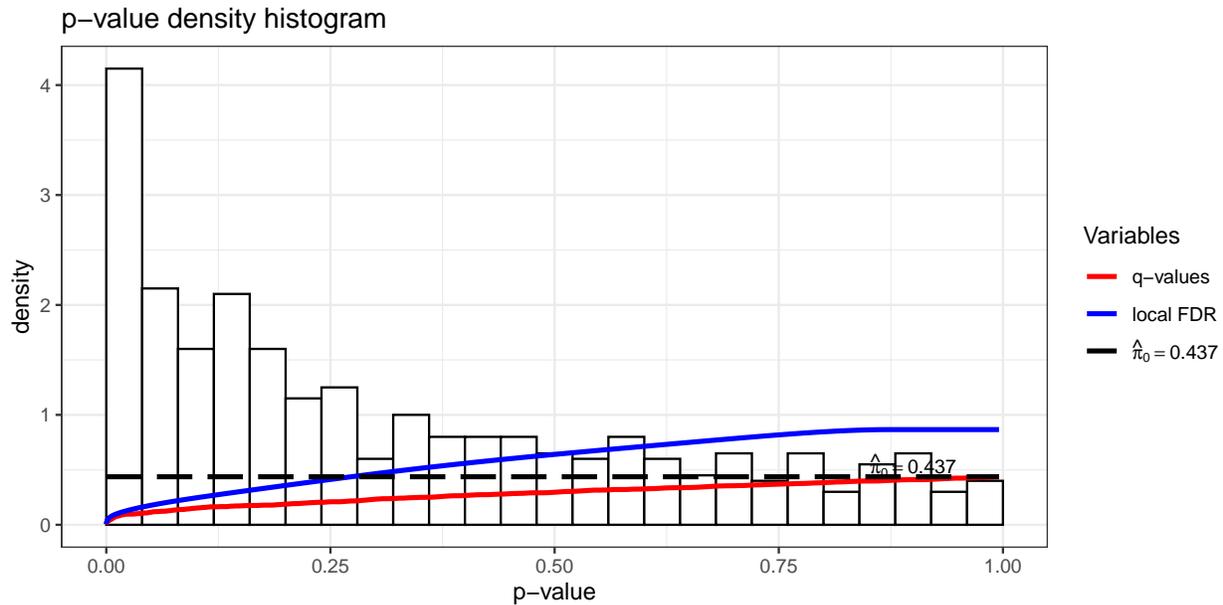
Figure 7: Applying the function `hist` to the slot `qvalueObj` in the `kidney` data set. Function is derived from the `qvalue` package.

```
fdr.level <- 0.1
sigGenes <- qvalues < fdr.level
```

View the qvalue vignette to get a more thorough discussion in how to use p-values, q-values, π_0 estimate and local false discovery rates to determine significant genes.

7   Case study: longitudinal time course experiment

In the longitudinal time course study, the goal is to identify genes that show "between-class temporal differential expression", i.e., genes that show statistically significant differences in expression over time between the various groups. The `endotoxin` data set provides gene expression measurements in an endotoxin study where four subjects were given endotoxin and four subjects were given a placebo. Blood samples were collected and leukocytes were isolated from the samples before infusion. Measurements were recorded at times 2, 4, 6, 9, 24 hours. We are interested in identifying genes that vary over time between the endotoxin and control groups. See Calvano et al. (2005) for more details regarding the `endotoxin` dataset.

7.1   Importing the data

To import the `endotoxin` data use the `data` function:

```
data(endotoxin)
endoexpr <- endotoxin$endoexpr
class <- endotoxin$class
ind <- endotoxin$ind
time <- endotoxin$time
```

There are a few covariates in this data set: `endoexpr`, `class`, `ind`, and `time`. There are 8 individuals in
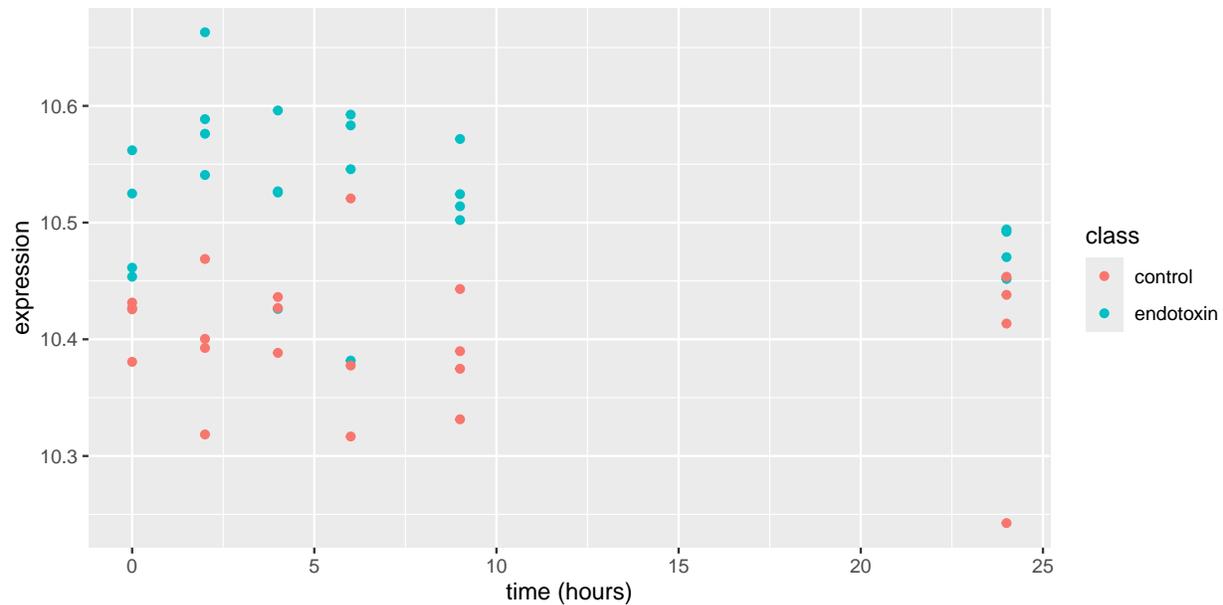
Figure 8: Plot of gene 2 in the `endotoxin` study.

the experiment (`ind`) that were sampled at multiple time points (`time`) that were either "endotoxin" or "control" (`class`). The `endoexpr` variable contains the expression values of the experiment.

To show an example gene, the expression values of the second gene are shown in Figure 8. It is very difficult to find a trend for this particular gene. Instead, we need to adjust the data with the models in the study.

7.2   Creating the full and null models

In order to find differentially expressed genes, there first needs to be an full and null model for the study. There are two ways to input the experimental models in `edge`: `build_models` and `build_study`. `build_study` should be used by users unfamiliar with formulating the full and null models but are familiar with the covariates in the study:

```
de_obj <- build_study(data = endoexpr, grp = class,
    tme = time, ind = ind, sampling = "timecourse")
```

`grp` is for the variable which group each individual belongs to, `tme` is the time variable, `ind` is used when individuals are sampling multiple times and `sampling` describes the type of experiment. Since `endotoxin` is a time course study, the `sampling` argument will be "timecourse". The `tme` variable will be the `time` variable, `ind` is the `individuals` variable and the `grp` variable is `class`. To view the models created by `build_study`:

```
fullModel(de_obj)

## ~grp + ns(tme, df = 2, intercept = FALSE) + (grp):ns(tme, df = 2,
##     intercept = FALSE)
## <environment: 0x55fd91242430>

nullModel(de_obj)
```

```
## ~grp + ns(tme, df = 2, intercept = FALSE)
## <environment: 0x55fd91242430>
```

See Storey et al. (2005) for how the models in the `endotoxin` experiment are formed. Alternatively, if the user is familiar with their full and null models in the study then `build_models` can be used to input the models directly:

```
cov <- data.frame(ind = ind, tme = time, grp = class)
null_model <- ~grp + ns(tme, df = 2, intercept = FALSE)
null_model <- ~grp + ns(tme, df = 2, intercept = FALSE) +
    (grp):ns(tme, df = 2, intercept = FALSE)
de_obj <- build_models(data = endoexpr, cov = cov,
    full.model = null_model, null.model = null_model)
```

The `cov` argument is a data frame of all the relevant covariates, `full.model` and `null.model` are the full and null models of the experiment, respectively. Notice that the models must be formatted as a formula and contain the same variable names as in the `cov` data frame. We are interested in testing whether the full model improves the model fit of a gene significantly when compared to the null model. If it does not, then we can conclude that there is no significant difference in this gene between the endotoxin and the control as time goes on.

The variable `de_obj` is an `deSet` object that stores all the relevant experimental data. The `deSet` object is discussed further in the next section.

7.3   The `deSet` object

Once either `build_models` or `build_study` is used, an `deSet` object is created. To view the slots contained in the object:

```
slotNames(de_obj)

##  [1] "null.model"        "full.model"
##  [3] "null.matrix"       "full.matrix"
##  [5] "individual"        "qvalueObj"
##  [7] "experimentData"    "assayData"
##  [9] "phenoData"         "featureData"
## [11] "annotation"        "protocolData"
## [13] ".__classVersion__"
```

A description of each slot is listed below:

- `full.model`: the full model of the experiment. Contains the biological variables of interest and the adjustment variables.

- `null.model`: the null model of the experiment. Contains the adjustment variables in the experiment.

- `full.matrix`: the full model in matrix form.

- `null.matrix`: the null model in matrix form.

- `individual`: variable that keeps track of individuals (if same individuals are sampled multiple times).

- `qvalueObj`: qvalue list. Contains p-values, q-values and local false discovery rates of the significance

analysis. See the `qvalue` package for more details.

- `ExpressionSet`: inherits the slots from `ExpressionSet` object.

`ExpressionSet` contains the expression measurements and other information from the experiment. The `deSet` object inherits all the functions from an `ExpressionSet` object. As an example, to access the expression values, one can use the function `exprs` or to access the covariates, `pData`:

```
gibexpr <- exprs(de_obj)
cov <- pData(de_obj)
```

The `ExpressionSet` class is a widely used object in Bioconductor and more information can be found here. See the section 10 on `ExpressionSet` to get a better understanding of how it integrates into the `edge` framework.

As an example of how to access the slots of `de_obj` suppose we are interested in viewing the full and null models. The models can be accessed by:

```
fullModel(de_obj)

## ~grp + ns(tme, df = 2, intercept = FALSE) + (grp):ns(tme, df = 2,
##      intercept = FALSE)

nullModel(de_obj)

## ~grp + ns(tme, df = 2, intercept = FALSE) + (grp):ns(tme, df = 2,
##      intercept = FALSE)
```

Next, we can extract the models in matrix form for computational analysis:

```
full_matrix <- fullMatrix(de_obj)
null_matrix <- nullMatrix(de_obj)
```

See `?deSet` for additional functions to access different slots of the `deSet` object.

7.4   Fitting the data

The `fit_models` function is an implementation of least squares using the full and null models:

```
ef_obj <- fit_models(de_obj, stat.type = "lrt")
```

The `stat.type` argument specifies whether you want the `odp` or `lrt` fitted values. The difference between choosing "odp" and "lrt" is that "odp" centers the data by the null model fit which is necessary for downstream analysis in the optimal discovery procedure. `fit_models` creates another object with the following slots:

- `fit.full`: fitted values from the full model.

- `fit.null`: fitted values from null model.

- `res.full`: residuals from the full model.

- `res.null`: residuals from the null model.

- `dH.full`: diagonal elements in the projection matrix for the full model.

24

Figure 9: Plot of gene 2 in the `endotoxin` study after applying the full and null model fit. The "raw" column is the expression values of the original data.

- `beta.coef`: the coefficients for the full model.

- `stat.type`: statistic type used, either "odp" or "lrt".

To access the fitted coefficients of the full model in `ef_obj`:

```
betaCoef(ef_obj)
```

To access the full and null residuals:

```
alt_res <- resFull(ef_obj)
null_res <- resNull(ef_obj)
```

To access the fitted values:

```
alt_fitted <- fitFull(ef_obj)
null_fitted <- fitNull(ef_obj)
```

See `?deFit` for more details on accessing the slots in an `deFit` object. The fitted values of the second gene are shown in Figure 9. It appears that the full model fits a pattern that might be observed in the raw data. Next, we have to test whether the observed differences between the model fits are significant.

7.5 Significance analysis

Interpreting the models in a hypothesis test is very intuitive: Does the full model better fit the data when compared to the null model? For the fitted values of the second gene plotted in Figure 9, it seems that the full model fits the data better than the null model. In order to conclude it is significant, we need to calculate the p-value. The user can use either the optimal discovery procedure or likelihood ratio test.

7.5.1 Likelihood ratio test

The `lrt` function performs a likelihood ratio test to determine p-values:

```
de_lrt <- lrt(de_obj, nullDistn = "normal")
```

If the null distribution, `nullDistn`, is calculated using "bootstrap" then residuals from the full model are re-sampled and added to the null model to simulate a distribution where there is no differential expression. Otherwise, the default input is "normal" and the assumption is that the null statistics follow a F-distribution. See `?lrt` for additional arguments.

7.5.2 Optimal discovery procedure

`odp` performs the optimal discovery procedure, which is a new approach developed by Storey et al. (2005) for optimally performing many hypothesis tests in a high-dimensional study. When testing a feature, information from all the features is utilized when testing for significance of a feature. It guarantees to maximize the number of expected true positive results for each fixed number of expected false positive results which is related to the false discovery rate.

The optimal discovery procedure can be implemented on `de_obj` by the `odp` function:

```
de_odp <- odp(de_obj, bs.its = 50, verbose = FALSE,
    n.mods = 50)
```

The number of bootstrap iterations is controlled by `bs.its`, `verbose` prints each bootstrap iteration number and `n.mods` is the number of clusters in the k-means algorithm.

A k-means algorithm is used to assign genes to groups in order to speed up the computational time of the algorithm. If `n.mods` is equal to the number of genes then the original optimal discovery procedure is used. Depending on the number of genes, this setting can take a very long time. Therefore, it is recommended to use a small `n.mods` value to substantially decrease the computational time. In Woo et al. (2011), it is shown that assigning `n.mods` to about 50 will cause a negligible loss in power. Type `?odp` for more details on the algorithm.

7.6 Significance results

The `summary` function can be used on an `deSet` object to give an overview of the analysis:

```
summary(de_odp)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 46 samples
##    element names: exprs
## protocolData: none
## phenoData
##    sampleNames: 1 2 ... 46 (46 total)
##    varLabels: tme grp
##    varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
```

```
##
## de Analysis Summary
##
## Total number of arrays: 46
## Total number of probes: 500
##
## Biological variables:
##  Null Model:~grp + ns(tme, df = 2, intercept = FALSE)
## <environment: 0x55fd9a008848>
##
##  Full Model:~grp + ns(tme, df = 2, intercept = FALSE) + (grp):ns(tme, df = 2,
##      intercept = FALSE)
## <environment: 0x55fd9a008848>
##
## Individuals:
##  [1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4
## [23] 4 4 5 5 5 5 5 5 6 6 6 6 7 7 7 7 7 7 8 8 8 8
## [45] 8 8
## Levels: 1 2 3 4 5 6 7 8
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.4581485
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value        8     28    65     91   112  142
## q-value        0      0    28     44    79  108
## local fdr      0      0    17     28    40   66
##             <1
## p-value    500
## q-value    500
## local fdr 500
```

There are three core summaries: `ExpressionSet` summary, `edge` analysis and statistical significance sum-mary. The `ExpressionSet` summary shows a summary of the `ExpressionSet` object. `edge` analysis shows an overview of the models used and other information about the data set. The significance analysis shows the proportion of null genes, π_0, and significant genes at various cutoffs in terms of p-values, q-values and local false discovery rates.

The function `qvalueObj` can be used on `de_odp` to extract the significance results:

```
sig_results <- qvalueObj(de_odp)
```

The object `sig_results` is a list with the following slots:

```
names(sig_results)
```

```
## [1] "call"       "pi0"        "qvalues"
```

Figure 10: Applying the function `hist` to the slot `qvalueObj` in the `endotoxin` data set. Function is derived from the `qvalue` package.

```
## [4] "pvalues"    "lfdr"       "pi0.lambda"
## [7] "lambda"     "pi0.smooth" "stat0"
## [10] "stat"
```

The key variables are `pi0`, `pvalues`, `lfdr` and `qvalues`. The `pi0` variable provides an estimate of the proportion of null p-values, `pvalues` are the p-values, `qvalues` are the estimated q-values and `lfdr` are the local false discovery rates. Using the function `hist` on `sig_results` will produce a p-value histogram along with the density curves of q-values and local false discovery rate values:

```
hist(sig_results)
```

The plot is shown in Figure 10. To extract the p-values, q-values, local false discovery rates and the π_0 estimate:

```
pvalues <- sig_results$pvalues
qvalues <- sig_results$qvalues
lfdr <- sig_results$lfdr
pi0 <- sig_results$pi0
```

Making significance decisions based on p-values in multiple hypothesis testings problems can lead to accepting a lot of false positives in the study. Instead, using q-values to determine significant genes is recommended because it controls the false discovery rate at a level `alpha`. Q-values measure the proportion of false positives incurred when calling a particular test significant. For example, to complete our analysis of gene 2 in this example, lets view the q-value estimate:

```
qvalues[2]
```

```
## [1] 0.2498489
```

So for this particular gene, the q-value is 0.2498489. If we consider a false discovery rate cutoff of 0.1 then this gene is not significant. Therefore, the observed differences observed in Figure 9 are not significant so this particular gene is not differentially expressed between `class` as time varies.

To get a list of all the significant genes at a false discovery rate cutoff of 0.1:

```
fdr.level <- 0.1
sigGenes <- qvalues < fdr.level
```

View the qvalue vignette to get a more thorough discussion in how to use p-values, q-values, π_0 estimate and local false discovery rates to determine significant genes.

8   `sva`: Surrogate variable analysis

The `sva` package is useful for removing batch effects or any unwanted variation in an experiment. It does this by forming surrogate variables to adjust for sources of unknown variation. Details on the algorithm can be found in Leek and Storey (2007). `edge` uses the `sva` package in the function `apply_sva`. Suppose we are working with the `kidney` data in 6, then the first step is to create an `deSet` object by either using `build_models` or `build_study`:

```
library(splines)
cov <- data.frame(sex = sex, age = age)
null_model <- ~sex
full_model <- ~sex + ns(age, df = 4)
de_obj <- build_models(data = kidexpr, cov = cov,
    full.model = full_model, null.model = null_model)
```

To find the surrogate variables and add them to the experimental models in `de_obj`, use the function `apply_sva`:

```
de_sva <- apply_sva(de_obj, n.sv = 3, B = 10)

## Number of significant surrogate variables is:  3
## Iteration (out of 10 ):1  2  3  4  5  6  7  8  9  10
```

`n.sv` is the number of surrogate variables and `B` is the number of bootstraps. See `?apply_sva` for additional arguments. To see the terms that have been added to the models:

```
fullModel(de_sva)

## ~SV1 + SV2 + SV3 + sex + ns(age, df = 4)
## <environment: 0x55fd9ac783e8>

nullModel(de_sva)

## ~SV1 + SV2 + SV3 + sex
## <environment: 0x55fd9ac783e8>
```

The variables SV1, SV2 and SV3 are the surrogate variables formed by `sva`. To access the surrogate variables:

```
cov <- pData(de_sva)
names(cov)
```

```
## [1] "sex" "age" "SV1" "SV2" "SV3"

surrogate.vars <- cov[, 3:ncol(cov)]
```

Now `odp` or `lrt` can be used as in previous examples:

```
de_odp <- odp(de_sva, bs.its = 50, verbose = FALSE)
de_lrt <- lrt(de_sva, verbose = FALSE)
summary(de_odp)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 72 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: sex age ... SV3 (5 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## de Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 500
##
## Biological variables:
##  Null Model:~SV1 + SV2 + SV3 + sex
## <environment: 0x55fd9ac783e8>
##
##  Full Model:~SV1 + SV2 + SV3 + sex + ns(age, df = 4)
## <environment: 0x55fd9ac783e8>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.2689166
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value        2      8    35     59    88  132
## q-value        0      0     3     27    42  129
## local fdr      0      0     3     13    27   65
##             <1
## p-value    500
## q-value    500
```

```
## local fdr 500
```

And to extract the π_0 estimate, q-values, local false discovery rates and p-values:

```
qval_obj <- qvalueObj(de_odp)
qvals <- qval_obj$qvalues
lfdr <- qval_obj$lfdr
pvals <- qval_obj$pvalues
pi0 <- qval_obj$pi0
```

9   qvalue: Estimate the q-values

After `odp` or `lrt` is used, the user may wish to change some parameters used when calculating the q-values. This can be done by using the `apply_qvalue` function. Lets review the analysis process for the `kidney` dataset in 6: create the full and null models and then run `odp` or `lrt` to get significance results. Applying these steps in the `kidney` dataset:

```
# create models
library(splines)
cov <- data.frame(sex = sex, age = age)
null_model <- ~sex
full_model <- ~sex + ns(age, df = 4)
de_obj <- build_models(data = kidexpr, cov = cov,
    full.model = full_model, null.model = null_model)
# run significance analysis
de_obj <- odp(de_obj, bs.its = 50, verbose = FALSE)
```

Suppose we wanted to estimate π_0 using the "bootstrap" method in **qvalue** (see qvalue vignette for more details):

```
old_pi0est <- qvalueObj(de_obj)$pi0
de_obj <- apply_qvalue(de_obj, pi0.method = "bootstrap")
new_pi0est <- qvalueObj(de_obj)$pi0
```

```
##   old_pi0est new_pi0est
## 1  0.3859501  0.4514286
```

In this case, there is a difference between using the "smoother" method and "bootstrap" method. See `apply_qvalue` for additional arguments.

10   Advanced topic: Using the ExpressionSet object

`edge` was designed for complementing `ExpressionSet` objects in significance analysis. The `deSet` inherits all the slots from an `ExpressionSet` object and adds vital slots for significance analysis. The rest of this section is for advanced users because it requires knowledge of full and null model creation. To begin, lets create an `ExpressionSet` object from the `kidney` dataset:

```
library(edge)
anon_df <- as(data.frame(age=age, sex=sex), "AnnotatedDataFrame")
exp_set <- ExpressionSet(assayData = kidexpr, phenoData = anon_df)
```

In the `kidney` experiment they were interested in finding the effect of age on gene expression. In this case,

we handle the time variable, `age`, by fitting a natural spline curve as done in Storey et al. (2005). The relevant models for the experiment can be written as

```r
library(splines)
null_model <- ~1 + sex
full_model <- ~1 + sex + ns(age, intercept = FALSE,
    df = 4)
```

where **null_model** is the null model and **full_model** is the full model. The **sex** covariate is an adjustment variable while **age** is the biological variable of interest. It is important to note that it is necessary to include the adjustment variables in the formulation of the full models as done above.

Having both **expSet** and the hypothesis models, the function **deSet** can be used to create an **deSet** object:

```r
de_obj <- deSet(exp_set, full.model = full_model,
    null.model = null_model)
slotNames(de_obj)

##  [1] "null.model"        "full.model"
##  [3] "null.matrix"       "full.matrix"
##  [5] "individual"        "qvalueObj"
##  [7] "experimentData"    "assayData"
##  [9] "phenoData"         "featureData"
## [11] "annotation"        "protocolData"
## [13] ".__classVersion__"
```

From the slot names, it is evident that the **deSet** object inherits the **ExpressionSet** slots in addition to other slots relating to the significance analysis. See section 6.3 for more details on the **deSet** slots. We can now simply run **odp** or **lrt** for significance results:

```r
de_odp <- odp(de_obj, bs.its = 50, verbose = FALSE)
de_lrt <- lrt(de_obj)
summary(de_odp)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 72 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: age sex
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## de Analysis Summary
##
## Total number of arrays: 72
```

```
## Total number of probes: 500
##
## Biological variables:
##   Null Model:~1 + sex
##
##   Full Model:~1 + sex + ns(age, intercept = FALSE, df = 4)
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.3658784
##
## Cumulative number of significant calls:
##
##            <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value         3      6    35     64   111  156
## q-value         0      0     3      9    35  128
## local fdr       0      0     3      6    26   58
##             <1
## p-value    500
## q-value    500
## local fdr  500
```

And use the function `qvalueObj` to extract the π_0 estimate, q-values, local false discovery rates and p-values:

```
qval_obj <- qvalueObj(de_odp)
qvals <- qval_obj$qvalues
lfdr <- qval_obj$lfdr
pvals <- qval_obj$pvalues
pi0 <- qval_obj$pi0
```

Acknowledgements

References

Andrew J Bass and John D Storey. The optimal discovery procedure for significance analysis of general gene expression studies. *Bioinformatics*, 37(3):367–374, 08 2020. doi: 10.1093/bioinformatics/btaa707. URL https://doi.org/10.1093/bioinformatics/btaa707.

SE Calvano, W Xiao, DR Richards, RM Felciano, HV Baker, RJ Cho, RO Chen, BH Brownstein, JP Cobb, SK Tschoeke, C Miller-Graziano, LL Moldawer, MN Mindrinos, RW Davis, RG Tompkins, and SF Lowry. A network-based analysis of systemic inflammation in humans. *Nature*, 437:1032–1037, 2005. doi: 10.1038/nature03985. URL http://www.nature.com/nature/journal/v437/n7061/full/nature03985.html.

Ingrid Hedenfalk, David Duggan, Yidong Chen, Michael Radmacher, Michael Bittner, Richard Simon, Paul Meltzer, Barry Gusterson, Manel Esteller, Mark Raffeld, Zohar Yakhini, Amir Ben-Dor, Edward

Dougherty, Juha Kononen, Lukas Bubendorf, Wilfrid Fehrle, Stefania Pittaluga, Sofia Gruvberger, Niklas Loman, Oskar Johannsson, Håkan Olsson, Benjamin Wilfond, Guido Sauter, Olli-P. Kallioniemi, Åke Borg, and Jeffrey Trent. Gene-expression profiles in hereditary breast cancer. *New England Journal of Medicine*, 344(8):539–548, 2001. doi: 10.1056/NEJM200102223440801. URL http://dx.doi.org/10.1056/NEJM200102223440801. PMID: 11207349.

Y Idaghdour, JD Storey, SJ Jadallah, and G Gibson. A genome-wide gene expression signature of environmental geography in leukocytes of moroccan amazighs. *PLoS Genetics*, 4. doi: 10.1371/journal.pgen. 1000052.

Jeffrey T Leek and John D Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genet*, 3(9):e161, 09 2007. doi: 10.1371/journal.pgen.0030161.

Jeffrey T Leek and John D Storey. A general framework for multiple testing dependence. *Proc Natl Acad Sci U S A*, 105(48):18718–23, Dec 2008. doi: 10.1073/pnas.0808709105.

Jeffrey T. Leek, Eva Monsen, Alan R. Dabney, and John D. Storey. Edge: extraction and analysis of differential gene expression. *Bioinformatics*, 22(4):507–508, 2006. doi: 10.1093/bioinformatics/btk005. URL http://bioinformatics.oxfordjournals.org/content/22/4/507.abstract.

Graham E. J Rodwell, Rebecca Sonu, Jacob M Zahn, James Lund, Julie Wilhelmy, Lingli Wang, Wenzhong Xiao, Michael Mindrinos, Emily Crane, Eran Segal, Bryan D Myers, James D Brooks, Ronald W Davis, John Higgins, Art B Owen, and Stuart K Kim. A transcriptional profile of aging in the human kidney. *PLoS Biol*, 2(12):e427, 11 2004. doi: 10.1371/journal.pbio.0020427.

JD Storey. A direct approach to false discovery rates. *Journal of the Royal Statistical Society Series B-Statistical Methodology*, 64:479–498, 2002.

John D. Storey. The optimal discovery procedure: a new approach to simultaneous significance testing. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(3):347–368, 2007. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2007.005592.x. URL http://dx.doi.org/10.1111/j.1467-9868.2007.005592.x.

John D Storey and Robert Tibshirani. Statistical significance for genomewide studies. *Proc Natl Acad Sci U S A*, 100(16):9440–5, Aug 2003. doi: 10.1073/pnas.1530509100.

John D. Storey, Wenzhong Xiao, Jeffrey T. Leek, Ronald G. Tompkins, and Ronald W. Davis. Significance analysis of time course microarray experiments. *Proceedings of the National Academy of Sciences of the United States of America*, 102(36):12837–12842, 2005. doi: 10.1073/pnas.0504609102. URL http://www.pnas.org/content/102/36/12837.abstract.

John D. Storey, James Y. Dai, and Jeffrey T. Leek. The optimal discovery procedure for large-scale significance testing, with applications to comparative microarray experiments. *Biostatistics*, 8(2):414–432, 2007. doi: 10.1093/biostatistics/kxl019. URL http://biostatistics.oxfordjournals.org/content/8/2/414.abstract.

Sangsoon Woo, Jeffrey T. Leek, and John D. Storey. A computationally efficient modular optimal discovery procedure. *Bioinformatics*, 27(4):509–515, 2011. doi: 10.1093/bioinformatics/btq701. URL http://bioinformatics.oxfordjournals.org/content/27/4/509.abstract.