edge:
Extraction of Differential Gene Expression
Version 0.99.0

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

[1]Princeton University
[2]John Hopkins University

January 21, 2015

Contents

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

1 Introduction

`edge` is a package for significance analysis of DNA micro-array experiments and is able to identify genes that are differentially expressed between two or more different biological conditions (e.g., healthy versus diseased tissue). `edge` performs significance analysis by using a new method developed by Storey (2007) called the optimal discovery procedure (ODP). Whereas previously existing 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 more intuitive, often times more powerful, approach to multiple hypothesis testing when compared to traditional methods. The improvements in power from using the optimal discovery procedure are substantial; Figure 1 shows a comparison between edge and five leading software packages based on the Hedenfalk et al. (2001) breast cancer expression study.

`edge` also implements strategies that have been specifically designed for time course experiments. Many things can go wrong when using methods that have been designed for static experiments, and even though some significance analysis packages allow for users to enter information about time points, Storey et al. (2005) developed a procedure that simplifies the modelling process for time course experiments. In addition to identifying differentially expressed genes in both static and time course studies, `edge` includes implementations of popular packages such as `snm`, `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, 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

[1] 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
Theory paper that introduces the optimal discovery procedure and shows that it maximizes the expected true postive results for each number of fixed false positive results. The optimality is closely related to the false discovery rate.

[2] 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
Dicusses various ways of estimating the ODP statistic with applications to microarray experiments.

[3] 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
Previous implementations of the ODP are computationally infeasible for a large number of hypothesis tests. This paper introduces a computationally efficient implementation of ODP that this package is based on.

[4] 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
A methodology for analyzing time course microarray data is introduced and applied to two time course studies on humans.

3 Getting help

Hopefully, most questions relating to the package will be answered in the vignette but to get a more detailed account of how to use the functions simply type within R:

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

Please contact the authors directly with any issues regarding bugs. Otherwise, any questions or problems implementing `edge` will most efficiently be addressed on the Bioconductor mailing list, http://stat.ethz.ch/mailman/listinfo/bioconductor.

4 Quick start guide

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

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

The `kidney` study is interested in determining differentially expressed genes in the kidney as it ages. The `age` variable is the age of the subjects and the `sex` variable is whether the subjects were 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 an `edgeSet` object: `edgeModel` or `edgeStudy`. If the experiment models are unknown to the user, `edgeStudy` can be used to create the models:

```
edgeObj <- edgeStudy(data = kidexpr, adj.var = sex,
    tme = age, sampling = "timecourse")
fullMod <- fullModel(edgeObj)
nullMod <- nullModel(edgeObj)
```

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

If the alternative and null models are known to the user then `edgeModel` can be used to make an `edgeSet` object:

```
library(splines)
# alternative and null models
cov <- data.frame(sex = sex, age = age)
```

```
null.model <- ~sex
full.model <- ~sex + ns(age, df = 4)

edgeObj <- edgeModel(data = kidexpr, cov = cov, nullMod = null.model,
    altMod = full.model)
```

The `cov` is a data frame of covariates, the `nullMod` is the null model and the `altMod` is the alternative model. The input `cov` is a data frame with the column names the same as the variables in the alternative and null models. See Storey et al. (2005) to see why a natural spline curve is fit to the time variable in the study.

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

```
# optimal discovery procedure
edgeODP <- odp(edgeObj, verbose = FALSE)
# likelihood ratio test
edgeLRT <- lrt(edgeObj)
```

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

```
qvalObj <- qvalueObj(edgeODP)
qvals <- qvalObj$qvalues
pvals <- qvalObj$pvalues
lfdr <- qvalObj$lfdr
pi0 <- qvalObj$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 `gibson` dataset 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.

5.1 Importing the data

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

```
data(gibson)
names(gibson)
```

```
## [1] "gender"   "location" "batch"    "gibexpr"
```

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 that individuals from "VILLAGE" are less expressed when compared to other lifestyles. We should stop short of that observation because the data needs to be adjusted with the experimental models. Before that, the alternative and null model of the study needs to be formulated which is discussed the next section.

5.2 Creating the alternative and null models

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

```
edgeObj <- edgeStudy(data = gibson$gibexpr, adj.var = cbind(gibson$gender,
    gibson$batch), grp = gibson$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 alternative and null models in the study then `edgeModel` can be used to input the models directly:

```
cov <- data.frame(Gender = gibson$gender, Batch = gibson$batch,
    Location = gibson$location)
null.model <- ~Gender + Batch
alt.model <- ~Gender + Batch + Location
edgeObj <- edgeModel(data = gibson$gibexpr, cov = cov,
    altMod = alt.model, nullMod = null.model)
```

The `cov` argument is a data frame of all the relevant covariates, `altMod` and `nullMod` are the alternative 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 alternative model includes the `location` variable. Therefore, we are interested in testing whether the alternative 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 between Moroccan Amazigh groups for this particular gene.

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

5.3 The edgeSet object

Once either edgeModel or edgeStudy is used, an edgeSet object is created. To view the slots contained in the object:

```
slotNames(edgeObj)
```

```
##  [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 alternative model of the experiment
- null.model: the null model of the experiment
- full.matrix: the alternative model in matrix form
- null.matrix: the null model in matrix form
- individual: variable that keeps track of individuals (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 edgeSet 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(edgeObj)
cov <- pData(edgeObj)
```

The ExpressionSet class is a widely used object in Bioconductor and more information can be found http://www.bioconductor.org/packages/2.14/bioc/html/Biobase.html. See the advanced section to get a better understanding of how ExpressionSet objects integrate into the edge framework.

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

```
fullModel(edgeObj)
```

```
## ~Gender + Batch + Location
```

```
nullModel(edgeObj)
```

```
## ~Gender + Batch
```

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

```
full.matrix <- fullMatrix(edgeObj)
null.matrix <- nullMatrix(edgeObj)
```

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

5.4 Fitting the data

The `edgeFit` function is an implementation of least squares using the alternative and null models:

```
efObj <- edgeFit(edgeObj, 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. `edgeFit` creates another object with the following slots:

- `fit.full`: fitted values from the alternative model
- `fit.null`: fitted values from null model
- `res.full`: residuals from the alternative 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
- `stat.type`: statistic type used, either "odp" or "lrt"

To access the fitted coefficients of the alternative model in `efObj`:

```
betaCoef(efObj)
```

To access the alternative and null residuals:

```
alt.res <- resFull(efObj)
null.res <- resNull(efObj)
```

To access the fitted values:

```
alt.fitted <- fitFull(efObj)
null.fitted <- fitNull(efObj)
```

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

5.5 Significance analysis

Interpreting the models in a hypothesis test is very intuitive: Does the alternative 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 alternative model fits the data better than the null model. In order to conclude it's 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:

```
edgeLRT <- lrt(edgeObj, nullDistn = "normal")
```

If the null distribution, `nullDistn`, is calculated using "bootstrap" then residuals from the alternative 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 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 guarentees to maximize the number of expected true positive results for each fixed number of expected false positive results which is related to FDR. The optimal discovery procedure can be implemented on an `edgeSet` object by the `odp` function:

```
edgeODP <- odp(edgeObj, bs.its = 30, 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.

`n.mods` controls the number of clusters in the k-means algorithm where genes are assigned to a cluster based on the Kullback-Leiber distance. 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. The number of `bs.its` iterations recommended is 100.

5.6 Significance results

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

```
summary(edgeODP)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
```

```
## assayData: 10177 features, 46 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 46 (46 total)
##   varLabels: Gender Batch Location
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## edge Analysis Summary
##
## Total number of arrays: 46
## Total number of probes: 10177
##
## Biological variables:
##  Null Model:~Gender + Batch
##
##  Full Model:~Gender + Batch + Location
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.2991622
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value      932   1679  2934   3783  4571 5487
## q-value      744   1362  2903   4117  5251 6863
## local fdr    492    847  1869   2500  3185 4194
##              <1
## p-value   10177
## q-value   10177
## local fdr  9530
```

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 dataset. 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 `edgeODP` to extract the significance results:

```
sig.results <- qvalueObj(edgeODP)
```

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"
```

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. See Figure 5.

The q-value controls the false discovery at a level `alpha` and so making significance decisions with q-values is recommended. 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:

```
qval <- sig.results$qvalues
qval[1]

## [1] 3.659477e-05
```

So at this particular gene, the q-value is $3.6e-5$. If we consider a false discovery cutoff rate of 0.01 then this gene is highly significant. Therefore, the observed differences observed in Figure 3 are indeed significant so this particular gene is differentially expressed at each location.

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

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

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.

Gene expression measurements from `kidney` samples 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 dataset.

6.1 Importing the data

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

```
data(kidney)
names(kidney)

## [1] "tissue"  "age"     "sex"     "kidexpr"
```

There are a few covariates in this data set: `sex`, `age`, `tissue`, `kidexpr` and `kidcov`. We will focus on the expression values of the cortex `tissue` samples:

```
sex <- kidney$sex[kidney$tissue == "c"]
age <- kidney$age[kidney$tissue == "c"]
kidexpr <- log(kidney$kidexpr[, kidney$tissue ==
    "c"] + 10)
```

The two main covariates of interest for this example 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, the expression values of the first gene are shown in Figure 6. 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.

6.2 Creating the alternative and null models

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

```
edgeObj <- edgeStudy(data = kidexpr, adj.var = sex,
    bio.var = age, sampling = "timecourse")
```

`adj.var` is for the adjustment variables, `bio.var` is the biological variable and `sampling` describes the type of experiment. Since `kidney` is a time course study, the `sampling` argument will be "timecourse". The `bio.var` variable will be the `age` variable and the adjustment variable is `sex`.

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

```
cov <- data.frame(sex = sex, age = age)
null.model <- ~sex
alt.model <- ~sex + ns(age, df = 4)
edgeKid <- edgeModel(data = kidexpr, cov = cov, altMod = alt.model,
    nullMod = null.model)

## Error in eval(expr, envir, enclos):  could not find function "ns"
```

The `cov` argument is a data frame of all the relevant covariates, `altMod` and `nullMod` are the alternative 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 alternative model includes the `age` variable. Therefore, we are interested in testing whether the alternative 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 as it ages in the kidney.

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

6.3 The edgeSet object

Once either edgeModel or edgeStudy is used, an edgeSet object is created. To view the slots contained in the object:

```
slotNames(edgeObj)
```

```
##  [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 alternative model of the experiment
- null.model: the null model of the experiment
- full.matrix: the alternative model in matrix form
- null.matrix: the null model in matrix form
- individual: variable that keeps track of individuals (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 edgeSet 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(edgeObj)
cov <- pData(edgeObj)
```

The ExpressionSet class is a widely used object in Bioconductor and more information can be found http://www.bioconductor.org/packages/2.14/bioc/html/Biobase.html. See the advanced section to get a better understanding of how ExpressionSet objects integrate into the edge framework.

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

```
fullModel(edgeObj)
```

```
## ~adj.var + bio.var
## <environment: 0x10e229c8>
```

```
nullModel(edgeObj)
```

```
## ~adj.var
```

```
## <environment: 0x10e229c8>
```

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

```
full.matrix <- fullMatrix(edgeObj)
null.matrix <- nullMatrix(edgeObj)
```

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

6.4 Fitting the data

The `edgeFit` function is an implementation of least squares using the alternative and null models:

```
efObj <- edgeFit(edgeObj, 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. `edgeFit` creates another object with the following slots:

- `fit.full`: fitted values from the alternative model
- `fit.null`: fitted values from null model
- `res.full`: residuals from the alternative 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
- `stat.type`: statistic type used, either "odp" or "lrt"

To access the fitted coefficients of the alternative model in `efObj`:

```
betaCoef(efObj)
```

To access the alternative and null residuals:

```
alt.res <- resFull(efObj)
null.res <- resNull(efObj)
```

To access the fitted values:

```
alt.fitted <- fitFull(efObj)
null.fitted <- fitNull(efObj)
```

See `?edgeFit` for more details on accessing the slots in an `edgeFit` object. The fitted values of the first gene is shown in Figure 7. The null model fit is the average expression. It appears that the alternative model fits a pattern that might be observed in the raw data. Next, we have to test whether the observed differences between models in Figure 7 is significant.

6.5 Significance analysis

Interpreting the models in a hypothesis test is very intuitive: Does the alternative 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 alternative model fits the data better than the null model. In order to conclude it's 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:

```
edgeLRT <- lrt(edgeObj, nullDistn = "normal")
```

If the null distribution, `nullDistn`, is calculated using "bootstrap" then residuals from the alternative 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 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 guarentees to maximize the number of expected true positive results for each fixed number of expected false positive results which is related to FDR. The optimal discovery procedure can be implemented on an `edgeSet` object by the `odp` function:

```
edgeODP <- odp(edgeObj, bs.its = 30, 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.

`n.mods` controls the number of clusters in the k-means algorithm where genes are assigned to a cluster based on the Kullback-Leiber distance. 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. The number of `bs.its` iterations recommended is 100.

6.6 Significance results

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

```
summary(edgeODP)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
```

```
## assayData: 34061 features, 72 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: adj.var bio.var
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## edge Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 34061
##
## Biological variables:
##  Null Model:~adj.var
## <environment: 0xbff83e8>
##
##  Full Model:~adj.var + bio.var
## <environment: 0xbff83e8>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.8493684
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value       37    222   948   1888  3160 5458
## q-value        0      0     0      4    19  129
## local fdr      0      0     0      2     9   62
##              <1
## p-value   34061
## q-value   34061
## local fdr 25405
```

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 dataset. 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 `edgeODP` to extract the significance results:

```
sig.results <- qvalueObj(edgeODP)
```

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"
```

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. See Figure 5.

The q-value controls the false discovery at a level `alpha` and so making significance decisions with q-values is recommended. 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:

```
qval <- sig.results$qvalues
qval[1]

## [1] 0.8150455
```

So at this particular gene, the q-value is $3.6e-5$. If we consider a false discovery cutoff rate of 0.01 then this gene is highly significant. Therefore, the observed differences observed in Figure 3 are indeed significant so this particular gene is differentially expressed at each location.

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

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

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` dataset 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.

7.1 Importing the data

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

```
data(endotoxin)
names(endotoxin)

## [1] "class"    "endoexpr" "ind"      "time"
```

There are a few covariates in this data set: `expr`, `class`, `individual`, and `time`. There are 8 individuals in the experiment (`ind`) that were sampled at multiple time points (`time`) that were either "endotoxin" or "control" (`class`). The `expr` variable contains the expression values of the experiment:

As an example, the expression values of the first gene are shown in Figure **??**. 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 alternative and null models

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

```
edgeEndo <- edgeStudy(data = endotoxin$endoexpr,
    grp = endotoxin$class, tme = endotoxin$time,
    ind = endotoxin$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`.

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

```
cov <- data.frame(ind = endotoxin$ind, tme = endotoxin$time,
    grp = endotoxin$class)
null.model <- ~grp + ns(tme, df = 2, intercept = FALSE)
alt.model <- ~grp + ns(tme, df = 2, intercept = FALSE) +
    (grp):ns(tme, df = 2, intercept = FALSE)
edgeEndo <- edgeModel(data = endotoxin$endoexpr,
    cov = cov, altMod = alt.model, nullMod = null.model)

## Error in eval(expr, envir, enclos):  could not find function "ns"
```

The `cov` argument is a data frame of all the relevant covariates, `altMod` and `nullMod` are the alternative 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 alternative 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 as it ages in the kidney.

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

7.3 The `edgeSet` object

Once either `edgeModel` or `edgeStudy` is used, an `edgeSet` object is created. To view the slots contained in the object:

```
slotNames(edgeEndo)
```

```
##  [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 alternative model of the experiment
- `null.model`: the null model of the experiment
- `full.matrix`: the alternative model in matrix form
- `null.matrix`: the null model in matrix form
- `individual`: variable that keeps track of individuals (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 `edgeSet` 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(edgeEndo)
cov <- pData(edgeEndo)
```

The `ExpressionSet` class is a widely used object in Bioconductor and more information can be found http://www.bioconductor.org/packages/2.14/bioc/html/Biobase.html. See the advanced section to get a better understanding of how `ExpressionSet` objects integrate into the `edge` framework.

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

```
fullModel(edgeEndo)
```

```
## ~grpendotoxin + ns(tme, df = 2, intercept = FALSE) + (grpendotoxin):ns(tme,
##     df = 2, intercept = FALSE)
## <environment: 0xa283ed0>
```

```
nullModel(edgeEndo)
```

```
## ~grpendotoxin + ns(tme, df = 2, intercept = FALSE)
## <environment: 0xa283ed0>
```

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

```
full.matrix <- fullMatrix(edgeEndo)
null.matrix <- nullMatrix(edgeEndo)
```

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

7.4 Fitting the data

The `edgeFit` function is an implementation of least squares using the alternative and null models:

```
efObj <- edgeFit(edgeEndo, 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. `edgeFit` creates another object with the following slots:

- `fit.full`: fitted values from the alternative model
- `fit.null`: fitted values from null model
- `res.full`: residuals from the alternative 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
- `stat.type`: statistic type used, either "odp" or "lrt"

To access the fitted coefficients of the alternative model in `efObj`:

```
betaCoef(efObj)
```

To access the alternative and null residuals:

```
alt.res <- resFull(efObj)
null.res <- resNull(efObj)
```

To access the fitted values:

```
alt.fitted <- fitFull(efObj)
null.fitted <- fitNull(efObj)
```

See `?edgeFit` for more details on accessing the slots in an `edgeFit` object. The fitted values of the first gene is shown in Figure 7. The null model fit is the average expression. It appears that the alternative model fits a pattern that might be observed in the raw data. Next, we have to test whether the observed differences between models in Figure **??** is significant.

7.5 Significance analysis

Interpreting the models in a hypothesis test is very intuitive: Does the alternative 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 alternative model fits the data better than the null model. In order to conclude it's 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:

```
edgeLRT <- lrt(edgeEndo, nullDistn = "normal")
```

If the null distribution, `nullDistn`, is calculated using "bootstrap" then residuals from the alternative 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 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 guarentees to maximize the number of expected true positive results for each fixed number of expected false positive results which is related to FDR. The optimal discovery procedure can be implemented on an `edgeSet` object by the `odp` function:

```
edgeODP <- odp(edgeEndo, bs.its = 30, 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.

`n.mods` controls the number of clusters in the k-means algorithm where genes are assigned to a cluster based on the Kullback-Leiber distance. 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. The number of `bs.its` iterations recommended is 100.

7.6 Significance results

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

```
summary(edgeODP)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
```

```
## assayData: 5000 features, 46 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 46 (46 total)
##   varLabels: tme grpendotoxin
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## edge Analysis Summary
##
## Total number of arrays: 46
## Total number of probes: 5000
##
## Biological variables:
##  Null Model:~grpendotoxin + ns(tme, df = 2, intercept = FALSE)
## <environment: 0x87b8038>
##
##  Full Model:~grpendotoxin + ns(tme, df = 2, intercept = FALSE) + (grpendotoxin):ns(tme,
##     df = 2, intercept = FALSE)
## <environment: 0x87b8038>
##
## 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: 0.5425514
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value      125    250   492    701   916 1282
## q-value        0     79   249    360   473  709
## local fdr      0     66   152    219   292  388
##             <1
## p-value   5000
## q-value   5000
## local fdr 5000
```

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 dataset. 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 `edgeODP` to extract the significance results:

```
sig.results <- qvalueObj(edgeODP)
```

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"
```

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. See Figure 5.

The q-value controls the false discovery at a level `alpha` and so making significance decisions with q-values is recommended. 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:

```
qval <- sig.results$qvalues
qval[1]

## [1] 0.2543239
```

So at this particular gene, the q-value is $3.6e-5$. If we consider a false discovery cutoff rate of 0.01 then this gene is highly significant. Therefore, the observed differences observed in Figure 3 are indeed significant so this particular gene is differentially expressed at each location.

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

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

8 Using the sva package

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 `edgeSVA`. Suppose we are working with the `kidney` data in 6, then the first step is to create an `edgeSet` object by either using `edgeModel` or `edgeStudy`:

```
library(splines)
cov <- data.frame(sex = sex, age = age)
null.model <- ~sex
alt.model <- ~sex + ns(age, df = 4)
edgeObj <- edgeModel(data = kidexpr, cov = cov, altMod = alt.model,
    nullMod = null.model)
```

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

```
newEdgeObj <- edgeSVA(edgeObj, 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 `edgeSVA` for additional arguments. To see the terms that have been added to the models:

```
fullModel(newEdgeObj)

## ~sex + ns(age, df = 4) + SV1 + SV2 + SV3
## <environment: 0xb81d7b0>

nullModel(newEdgeObj)

## ~sex + SV1 + SV2 + SV3
## <environment: 0xb81d7b0>
```

The variables SV1, SV2 and SV3 are the surrogate variables formed by `sva`. Now `odp` can be used as in previous examples:

```
edgeODP <- odp(newEdgeObj, verbose = FALSE)

summary(edgeODP)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 34061 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:
##
## edge Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 34061
##
## Biological variables:
##  Null Model:~sex + SV1 + SV2 + SV3
## <environment: 0xb81d7b0>
##
##  Full Model:~sex + ns(age, df = 4) + SV1 + SV2 + SV3
## <environment: 0xb81d7b0>
```

```
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.6331189
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value      142    451  1712   3041  4773 7463
## q-value        0      9   111    232   491 1316
## local fdr      0      0    64    127   245  648
##              <1
## p-value   34061
## q-value   34061
## local fdr 32302
```

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

```
qvalObj <- qvalueObj(edgeODP)
qvals <- qvalObj$qvalues
lfdr <- qvalObj$lfdr
pvals <- qvalObj$pvalues
```

9 Using the snm package

The `snm` package allows for supervised normalization of microarrays on a gene expression matrix. Details on the algorithm can be found in Mecham et al. (2010). The `snm` package is implemented in the `edgeSNM` function. Continuing the analysis on the `kidney` study in 6:

```
# create models
edgeObj <- edgeStudy(data = kidexpr, adj.var = sex,
    bio.var = age, sampling = "timecourse")
```

Now that we have created `edgeObj`, we can adjust for additional array effects, dye effects and other intensity-dependent effects:

```
int.var <- data.frame(array.effects = as.factor(1:72))
edgeObj <- edgeSNM(edgeObj, int.var = int.var, diagnose = FALSE,
    verbose = FALSE)
```

The `int.var` is where the data frame of intensity-dependent effects are inputed, `diagnose` is a flag to let the software know whether to produce diagnostic plots. Additional arguments can be found by typing `?edgeSNM`. The authors of `snm` recommend that the probe level data be analyzed on a log-transformed scale and that the normalized data be inspected for latent structure using SVA 8.

Once the data has been normalized, `odp` can be used for get significance results:

```
edgeODP <- odp(edgeObj, verbose = FALSE)
```

```
summary(edgeODP)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 34061 features, 72 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: adj.var bio.var
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## edge Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 34061
##
## Biological variables:
##  Null Model:~adj.var
## <environment: 0xc91e6b8>
##
##  Full Model:~adj.var + bio.var
## <environment: 0xc91e6b8>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.7885024
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value       63    329  1326   2433  3891 6360
## q-value        0      0     3     10   151  429
## local fdr      0      0     3      5    84  235
##              <1
## p-value   34061
## q-value   34061
## local fdr 29182
```

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

```
qvalObj <- qvalueObj(edgeODP)
qvals <- qvalObj$qvalues
lfdr <- qvalObj$lfdr
pvals <- qvalObj$pvalues
```

10 Using the qvalue package

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 `edgeQvalue` function. Lets review the analysis process for the `kidney` dataset in 6: create the alternative and null models and then run `odp` or `lrt` to get significance results. Applying these steps in the `kidney` dataset:

```
# create models
edgeObj <- edgeStudy(data = kidexpr, adj.var = sex,
    bio.var = age, sampling = "timecourse")
# run significance analysis
edgeObj <- odp(edgeObj, verbose = FALSE)
```

Suppose we wanted to estimate pi_0 using the "bootstrap" method in `qvalue` (see `qvalue` vignette for more details):

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

```
##   old_pi0est new_pi0est
## 1  0.7885024  0.7928632
```

In this case, there is negligible difference between using the "smoother" method and "bootstrap" method but the point is that the arguments from the `qvalue` package can be passed through `edgeQvalue`. See `edgeQvalue` for additional information.

11 Figures

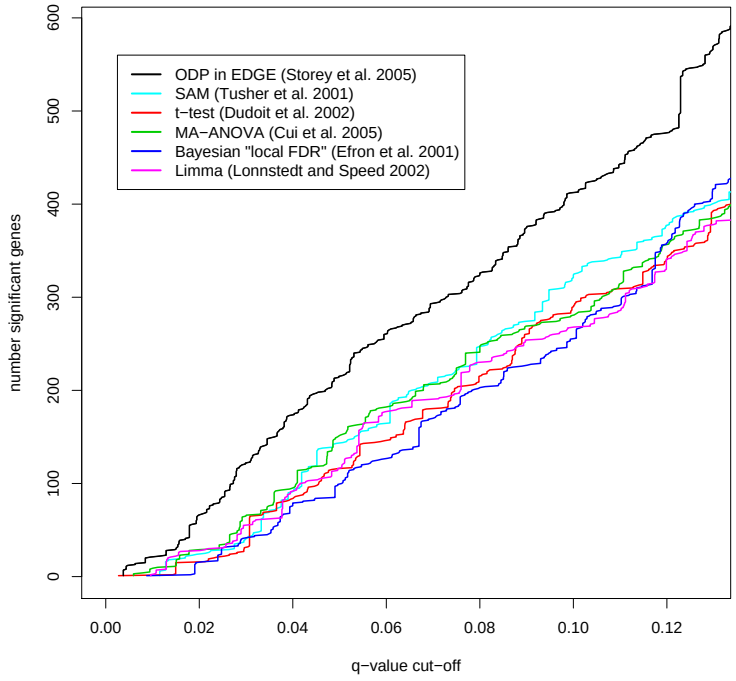

Figure 1: Comparison of EDGE to various other leading methods for identifying differential expressed genes in the Hedenfalk et al. (2001) study. Figure is from Leek et al. (2006).

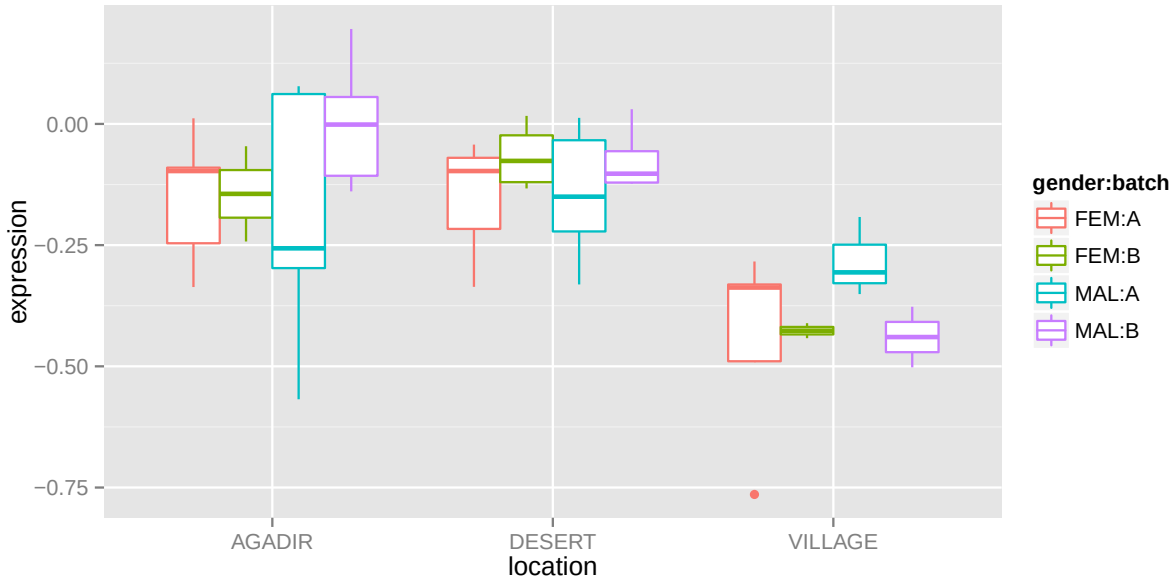

Figure 2: Plot of the first gene in the `gibson` study.

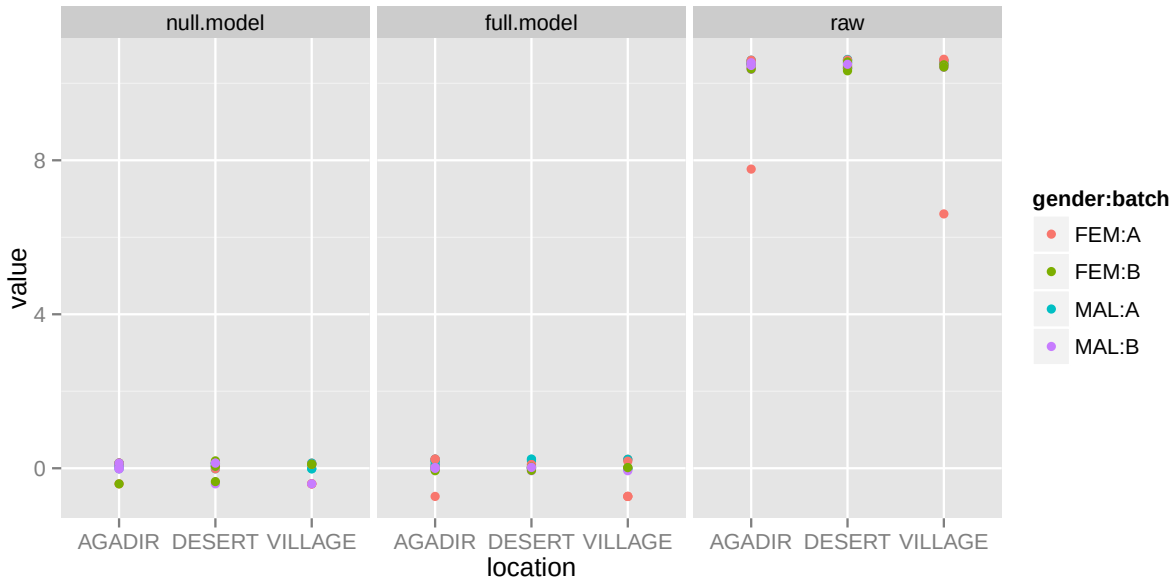

Figure 3: Plot of the first gene in the `gibson` study after applying the alternative and null model fit. The "raw" column is the expression values of the original data.

```
plot(sig.results)
```

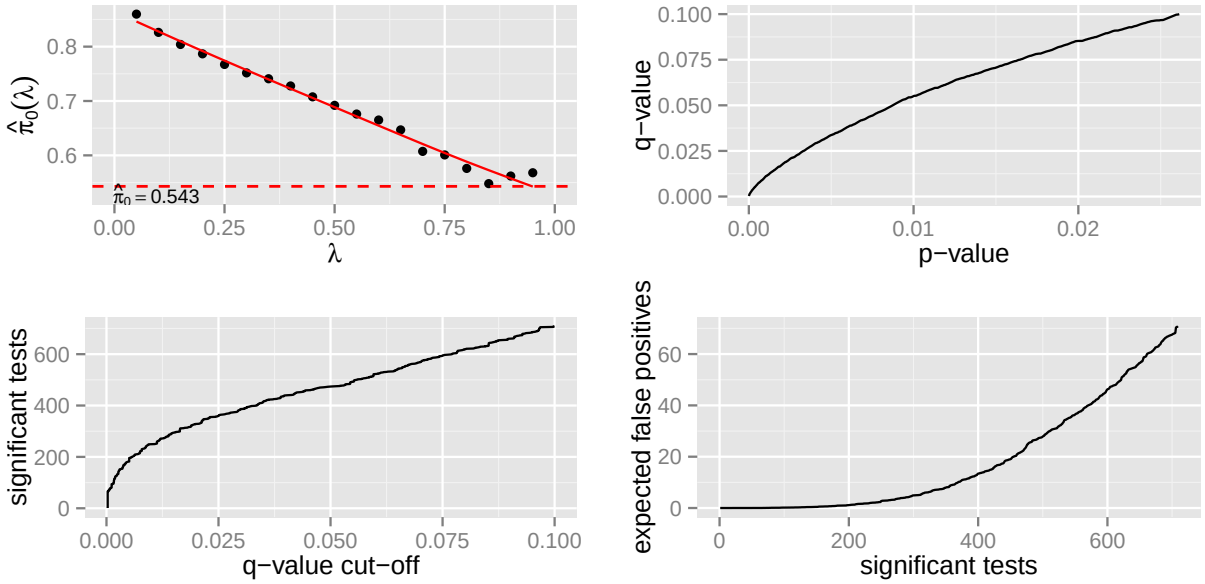

Figure 4: Apply the function `plot` to the slot `qvalueObj`. Function is derived from the `qvalue` package.

```
hist(sig.results)
```

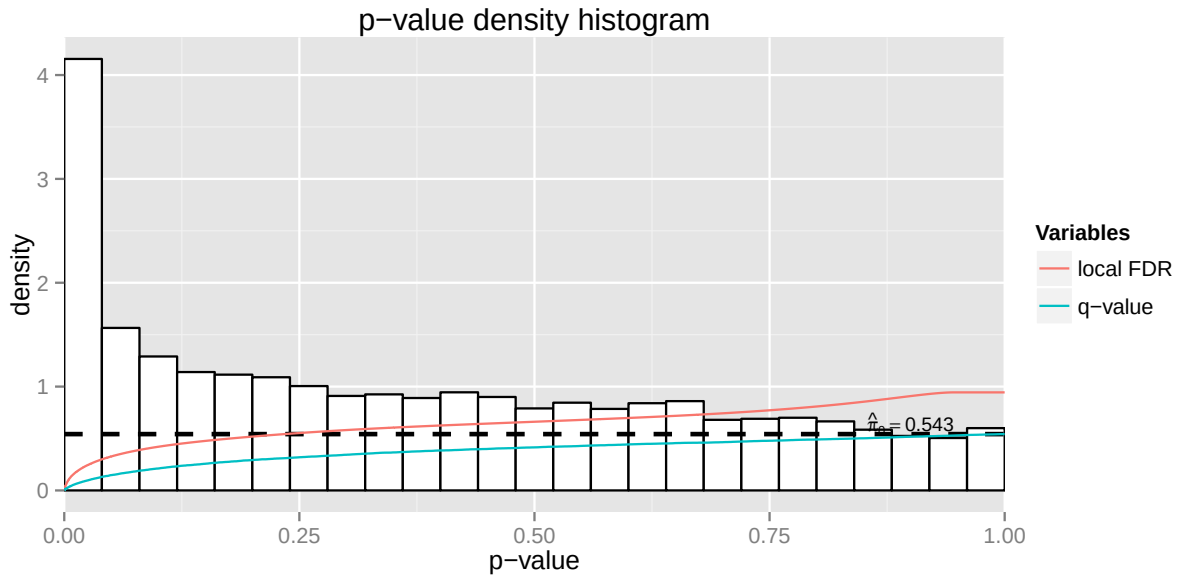

Figure 5: Apply the function `hist` to the slot `qvalueObj`. Function is derived from the `qvalue` package.

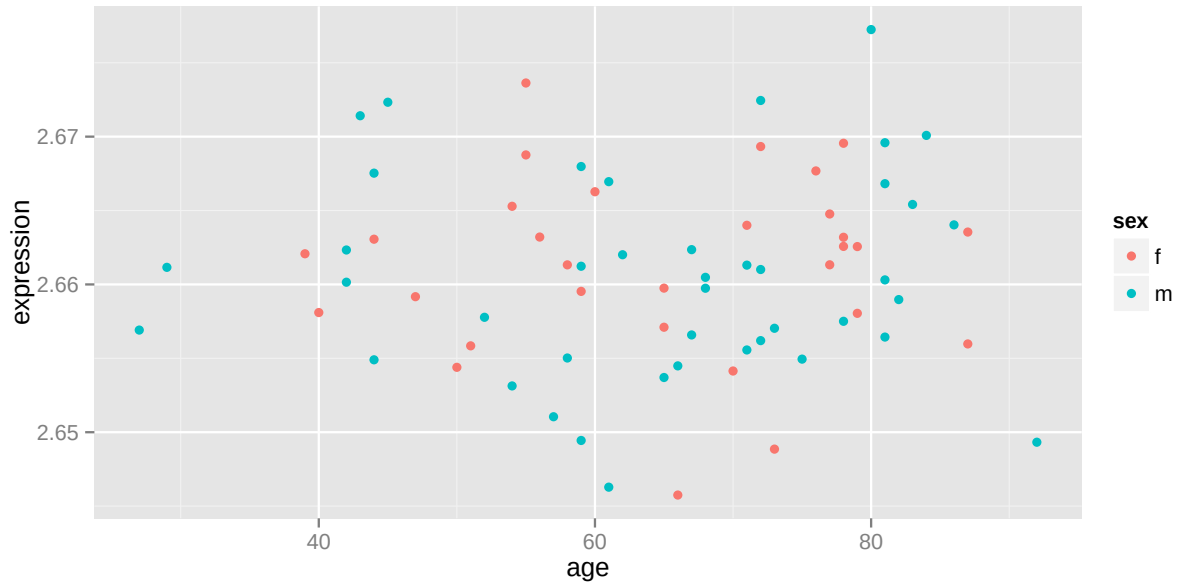

Figure 6: Plot of the first gene in the `kidney` study.

```
## Error in data.frame(age = age, sex = sex, null.model = fitVals0[1, ], :  arguments
                        imply differing number of rows:  72, 46
                  ## Error:  id variables not found in data:  age, sex
      ## Error:  Aesthetics must either be length one, or the same length as the
                                  dataProblems:age, sex
```

Figure 7: Plot of the first gene in the `gibson` study after applying the alternative and null model fit. The "raw" column is the expression values of the original data.

Acknowledgements

This software development has been supported in part by funding from the National Institutes of Health and the Office of Naval Research.

References

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, 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.

Brigham H. Mecham, Peter S. Nelson, and John D. Storey. Supervised normalization of microarrays. *Bioinformatics*, 26(10):1308–1315, 2010. doi: 10.1093/bioinformatics/btq118. URL http://bioinformatics.oxfordjournals.org/content/26/10/1308.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, 11 2004.

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, 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.