edge:
Extraction of Differential Gene Expression
Version 2.8.0

John D. Storey∗1,2, Jeffrey T. Leek3, and Andrew J. Bass1

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

April 24, 2017

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 .................................................. 3

2 Citing this package ........................................... 4

3 Getting help .................................................... 5

4 Quick start guide ............................................... 5

5 Case study: static experiment ............................... 7
   5.1 Importing the data ........................................ 7
   5.2 Creating the full and null models .......................... 7
   5.3 The deSet object ........................................... 9
   5.4 Fitting the data ............................................ 10
   5.5 Significance analysis ....................................... 11
      5.5.1 Likelihood ratio test ................................ 11
      5.5.2 Optimal discovery procedure ........................ 11
   5.6 Significance results ....................................... 12

6 Case study: independent time course experiment .......... 14
   6.1 Importing the data ........................................ 15
   6.2 Creating the full and null models .......................... 15
   6.3 The deSet object ........................................... 16
   6.4 Fitting the data ............................................ 17
   6.5 Significance analysis ....................................... 19
      6.5.1 Likelihood ratio test ................................ 19
      6.5.2 Optimal discovery procedure ........................ 19
   6.6 Significance results ....................................... 20

7 Case study: longitudinal time course experiment ........ 22
   7.1 Importing the data ........................................ 22
   7.2 Creating the full and null models .......................... 23
   7.3 The deSet object ........................................... 24
   7.4 Fitting the data ............................................ 25
   7.5 Significance analysis ....................................... 27
      7.5.1 Likelihood ratio test ................................ 27
      7.5.2 Optimal discovery procedure ........................ 27
   7.6 Significance results ....................................... 27

8 sva: Surrogate variable analysis ............................ 30

9 snm: Supervised normalization of microarray data ......... 32

10 qvalue: Estimate the q-values ............................... 34

11 Advanced topic: Using the ExpressionSet object ......... 34
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 snm, 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.

![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).](image)

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 snm, sva and qvalue to help simplify the analysis process for researchers.
The rest of the document details how to use \texttt{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:


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


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


When reporting results involving surrogate variable analysis (\texttt{apply_sva}), please cite:


When reporting results involving supervised normalization of microarrays (\texttt{apply_snm}), please cite:

Brigham H. Mecham, Peter S. Nelson, and John D. Storey. Supervised normalization of microarrays. \textit{Bioin-
To cite the edge R package itself, please type the following to retrieve the citation:

```r
citation("edge")
```

```latex
## To cite package 'edge' in publications
## use:
##
## John D. Storey, Jeffrey T. Leek and
## Andrew J. Bass (2015). edge: Extraction
## of Differential Gene Expression. R
## package version 2.8.0.
## 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 = {2015},
## note = {R package version 2.8.0},
## 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

```r
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:
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:

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

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 a `deSet` object:

```r
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:

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

To access the $\pi_0$ estimate, p-values, q-values and local false discovery rates for each gene, use the function `qvalueObj`:

```r
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.

5.1 Importing the data

To import the gibson data use the data function:

```r
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:

```r
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:

```r
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:

```r
slotNames(de_obj)
```

```r
## [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`:

```r
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 11 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:

```r
fullModel(de_obj)
```

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

```r
nullModel(de_obj)
```

```r
## ~Gender + Batch + Location
```
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.

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

```r
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:

```r
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.

• **stat.type**: statistic type used, either “odp” or “lrt”.

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

```r
betaCoef(ef_obj)
```

To access the full and null residuals:

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

To access the fitted values:

```r
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:

```r
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:

```r
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:

```r
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:
##
## de Analysis Summary
##
## Total number of arrays: 46
## Total number of probes: 500
##
## Biological variables:
## Null Model:~gender + batch
## <environment: 0x823f080>
##
## Full Model:~gender + batch + grp
## <environment: 0x823f080>
##
## .......
##
## Statistical significance summary:
## pi0: 0.338502
##
## Cumulative number of significant calls:
##
## | p-value    | 62   | 89   | 141  | 180  | 213  | 249  |
##|------------|------|------|------|------|------|------|
##| <0.001     | 89   | 141  | 180  | 213  | 249  |      |
##| <0.01      | 141  | 180  | 213  | 249  |      |      |
##| <0.025     | 180  | 213  | 249  |      |      |      |
##| <0.05      | 213  | 249  |      |      |      |      |
##| <0.1       | 249  |      |      |      |      |      |
##| <1         | 249  |      |      |      |      |      |

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, $\pi_0$, 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)
```
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:

```r
hist(sig_results)
```

The plot is shown in Figure 4. To extract the p-values, q-values, local false discovery rates and the π₀ estimate:

```r
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:

```r
qvalues[1]
## [1] 0.02409987
```

So for this particular gene, the q-value is 0.0240999. 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:

```r
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, π₀ 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:

```r
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:

```r
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:

```r
fullModel(de_obj)
## ~adj.var + ns(tme, df = 4, intercept = FALSE)
## <environment: 0xe61e888>
nullModel(de_obj)
## ~adj.var
## <environment: 0xe61e888>
```

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:

```r
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.

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:

```r
slotNames(de_obj)
## [1] "null.model"  "full.model"
## [3] "null.matrix"  "full.matrix"
## [5] "individual"  "qvalueObj"
## [7] "experimentData"  "assayData"
```
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`:

```r
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 11 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:

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

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

```r
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:

```r
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.
- 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:

```r
delrt <- 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:

```r
dedp <- 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:

```r
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: 0x1248fa10>
##
## Full Model:~adj.var + ns(tme, df = 4, intercept = FALSE)
## <environment: 0x1248fa10>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.4287065
##
## Cumulative number of significant calls:
##
##       <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value   4    6   25   53   84  145
## q-value   0    0   4    6   18  70
## local fdr 0    0   4    5   9   25
##
## <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, $\pi_0$, 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:

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

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

```r
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:

```r
hist(sig_results)
```

The plot is shown in Figure 7. To extract the p-values, q-values, local false discovery rates and the $\pi_0$ estimate:

```r
pvalues <- sig_results$pvalues
qvalues <- sig_results$qvalues
```
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 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, let's view the q-value estimate:

```r
qvalues[5]
## [1] 0.1707305
```

So for this particular gene, the q-value is 0.1707305. 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:

```r
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, $\pi_0$ 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:

```r
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 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 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:

```r
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`:

```r
fullModel(de_obj)
```

```r
## ~grp + ns(tme, df = 2, intercept = FALSE) + (grp):ns(tme, df = 2, 
##     intercept = FALSE)
## <environment: 0x11b467b0>
```

```r
nullModel(de_obj)
```

```r
## <environment: 0x11b467b0>
```
## \`grp + ns(tme, df = 2, intercept = FALSE)\`
## \langle\text{environment: 0x11b467b0}\rangle

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:

```r
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:

```r
slotNames(de_obj)
```

```r
## [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`:

```r
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 11 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:

```r
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:

```r
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:

```r
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.
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.

- **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 full model in `ef_obj`:

```r
betaCoef(ef_obj)
```

To access the full and null residuals:

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

To access the fitted values:

```r
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:

```r
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:

```r
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:

```r
summary(de_odp)
```

##
## ExpressionSet Summary
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, $\pi_0$, 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:

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

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

```r
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:

```r
hist(sig_results)
```

The plot is shown in Figure 10. To extract the p-values, q-values, local false discovery rates and the $\pi_0$
estimate:

```r
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 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, let's view the q-value estimate:

qvalues[2]
## [1] 0.2017277

So for this particular gene, the q-value is 0.2017277. 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, \( \pi_0 \) 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 a deSet object by either using build_models or build_study:

```r
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:

```r
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:
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:

dp_odp <- odp(de_sva, bs.its = 50, verbose = FALSE)
dp_lrt <- lrt(de_sva, verbose = FALSE)
summary(dp_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: 0x11fd8190>
## Full Model:SV1 + SV2 + SV3 + sex + ns(age, df = 4)
## <environment: 0x11fd8190>
##
And to extract the $\pi_0$ estimate, q-values, local false discovery rates and p-values:

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

9 **snm: Supervised normalization of microarray data**

There has been a lot of work done on separating signal from confounding factors, but a lot of algorithms fail to consider both the models of the study and the technical factors such as batch or array processing date. The **snm** package allows for supervised normalization of microarrays on a gene expression matrix. It takes into account both the experimental models and other technical factors in the experiments. Details on the algorithm can be found in Mecham et al. (2010). The **snm** package is implemented in the **apply.snm** function. Continuing the analysis on the **kidney** study in 6:

```r
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)
```

Now that we have created **de.obj**, we can adjust for additional array effects, dye effects and other intensity-dependent effects. In this example, we created array effects that are not existent in the real data set in order to show how to use the function:

```r
int.var <- data.frame(array.effects = as.factor(1:72))
de_snm <- apply_snm(de_obj, int.var = int.var, num.iter = 2,
     diagnose = FALSE, verbose = FALSE)
```

The **int.var** is where the data frame of intensity-dependent effects are inputted, **diagnose** is a flag to
let the software know whether to produce diagnostic plots. Additional arguments can be found by typing `?apply_snm`.

Once the data has been normalized, we can access the normalized matrix by using `exprs`:

```r
norm.matrix <- exprs(de_obj)
```

To run the significance analysis, `odp` or `lrt` can be used:

```r
de_odp <- odp(de_snm, bs.its = 50, 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
##   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:`sex`
##
##   Full Model:`sex + ns(age, df = 4)`
##
## .......
##
##
## Statistical significance summary:
##   pi0: 0.322866
##
## Cumulative number of significant calls:
##
##   <1e-04  <0.001  <0.01  <0.025  <0.05  <0.1
##   p-value  4    9    31    56    97   144
##   q-value  0    0     8    14    27  131
##   local fdr 0    0     4    9   17   62
##   <1
##   p-value  500
##   q-value  500
##   local fdr 500
```
And to extract the $\pi_0$ estimate, q-values, local false discovery rates and p-values:

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

### 10 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. Let's 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:

```r
# 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 $\pi_0$ using the “bootstrap” method in `qvalue` (see `qvalue` vignette for more details):

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

<table>
<thead>
<tr>
<th></th>
<th>old_pi0est</th>
<th>new_pi0est</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4457415</td>
<td>0.4914286</td>
</tr>
</tbody>
</table>

In this case, there is a difference between using the “smoother” method and “bootstrap” method. See `apply.qvalue` for additional arguments.

### 11 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, let's create an `ExpressionSet` object from the `kidney` dataset:

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

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
dep_odp <- odp(de_obj, bs.its = 50, verbose = FALSE)
dep_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: 0.414284
##
## Cumulative number of significant calls:
##
## | p-value | 500 | 500 | <0.05 | 0.1 |
## |---------|-----|-----|-------|-----|
## | q-value | 0   | 0   | 2     | 4   |
## | local fdr | 0 | 0 | 2 | 4 |

And use the function `qvalueObj` to extract the $\pi_0$ estimate, q-values, local false discovery rates and p-values:

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

**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


