edge:
Extraction of Differential Gene Expression
Version 2.6.0

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

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

October 17, 2016

First edition: October 2005

Most recent edition: March 2015

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

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

Contents

1 Introduction

The `edge` package implements methods for carrying out differential expression analyses of genome-wide gene expression studies. Significance testing using the optimal discovery procedure and generalized likelihood ratio test (equivalent to F-tests and t-tests) are implemented for general study designs. Special functions are available to facilitate the analysis of common study designs, including time course experiments. Other packages such as `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 **?** 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. **?** 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 **??** shows a comparison between `edge` and five leading software packages based on the **?** breast cancer expression study.

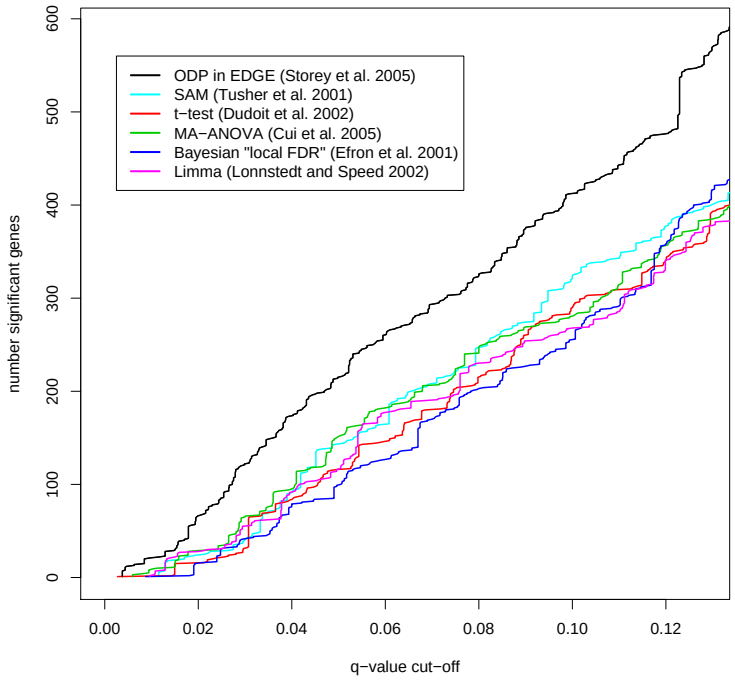

Figure 1: Comparison of EDGE to various other leading methods for identifying differential expressed genes in the **?** study. This figure is from **?**.

`edge` also implements strategies that have been specifically designed for time course experiments and other complex study designs. Specifically, **?** 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 `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 **?** methodology for time course experiments, see section **??**.

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 (`apply_sva`), please cite:

When reporting results involving supervised normalization of microarrays (`apply_snm`), please cite:

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

```
citation("edge")

##
## 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 2.6.0.
##   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 2.6.0},
##     url = {https://github.com/jdstorey/edge},
##   }
##
## ATTENTION: This citation information has
## been auto-generated from the package
## DESCRIPTION file and may need manual
## editing, see 'help("citation")'.
```

3 Getting help

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

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

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

4 Quick start guide

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Case study: static experiment

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

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 **?** for additional information regarding the data.

5.1 Importing the data

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

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

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

As an example, the expression values of the first gene are shown in Figure **??**. In the figure, it appears that

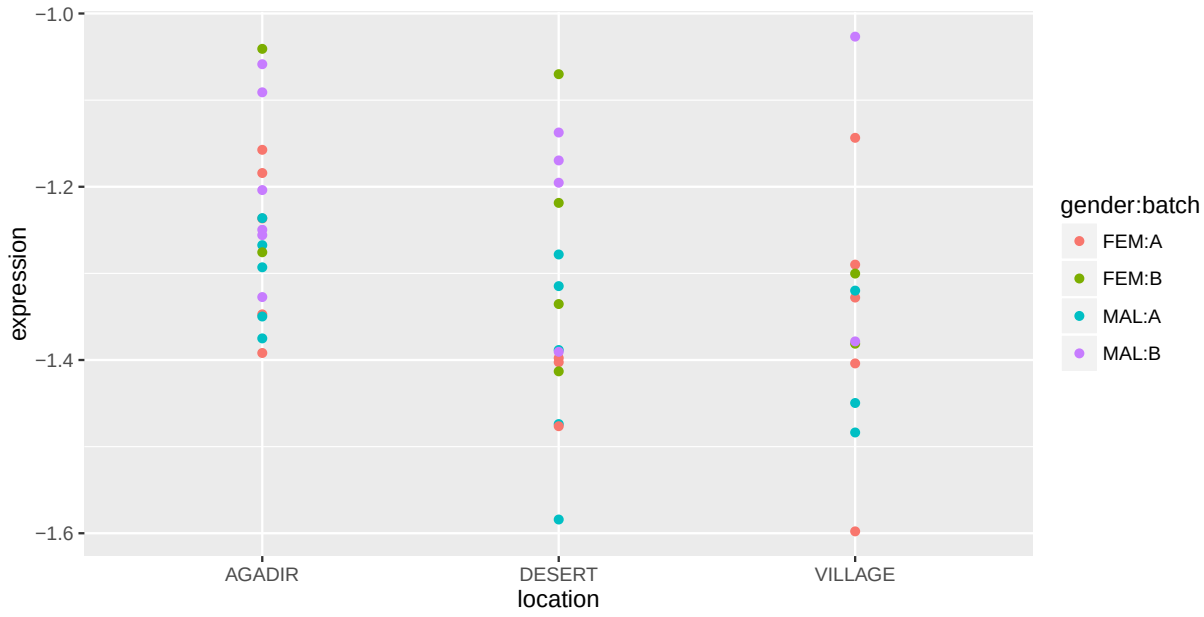

Figure 2: Plot of gene 1 in the `gibson` study.

the individuals from "VILLAGE" are more expressed when compared to the other lifestyles. We should stop short of that observation because the data needs to be adjusted with the experimental models. Before that, the full and null models of the study needs to be carefully formulated.

5.2 Creating the full and null models

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

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

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

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

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

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

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

5.3 The `deSet` object

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

```
slotNames(de_obj)
```

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

A description of each slot is listed below:

- `full.model`: the full model of the experiment. Contains the biological variables of interest and the adjustment variables.
- `null.model`: the null model of the experiment. Contains the adjustment variables in the experiment.
- `full.matrix`: the full model in matrix form.
- `null.matrix`: the null model in matrix form.
- `individual`: variable that keeps track of individuals (if same individuals are sampled multiple times).
- `qvalueObj`: `qvalue` list. Contains p-values, q-values and local false discovery rates of the significance analysis. See the `qvalue` package for more details.
- `ExpressionSet`: inherits the slots from `ExpressionSet` object.

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

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

The `ExpressionSet` class is a widely used object in Bioconductor and more information can be found here.

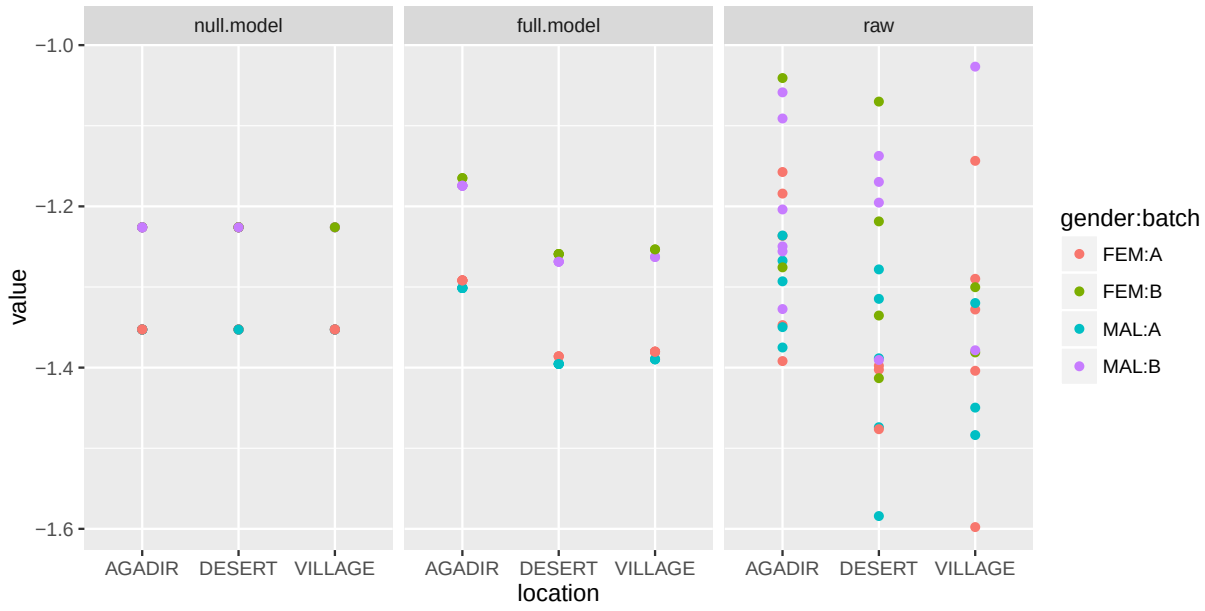

Figure 3: Plot of gene 1 in the `gibson` study after applying the full and null model fit. The "raw" column are the expression values of the original data.

See the section **??** on `ExpressionSet` to get a better understanding of how it integrates into the `edge` framework.

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

```
fullModel(de_obj)

## ~Gender + Batch + Location

nullModel(de_obj)

## ~Gender + Batch + Location
```

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

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

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

5.4 Fitting the data

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

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

The `stat.type` argument specifies whether you want the `odp` or `lrt` fitted values. The difference between

choosing "odp" and "lrt" is that "odp" centers the data by the null model fit which is necessary for downstream analysis in the optimal discovery procedure. `fit_models` creates another object with the following slots:

- `fit.full`: fitted values from the full model.
- `fit.null`: fitted values from null model.
- `res.full`: residuals from the full model.
- `res.null`: residuals from the null model.
- `dH.full`: diagonal elements in the projection matrix for the full model.
- `beta.coef`: the coefficients for the full model.
- `stat.type`: statistic type used, either "odp" or "lrt".

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

```
betaCoef(ef_obj)
```

To access the full and null residuals:

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

To access the fitted values:

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

See `?deFit` for more details on accessing the slots in an `deFit` object. The fitted values of the first gene are shown in Figure **??**. 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 **??**, it seems that the full model fits the data better than the null model. In order to conclude that it is significant, we need to calculate the p-value. The user can use either the optimal discovery procedure or likelihood ratio test.

5.5.1 Likelihood ratio test

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

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

If the null distribution, `nullDistn`, is calculated using "bootstrap" then residuals from the full model are re-sampled and added to the null model to simulate a distribution where there is no differential expression.

Otherwise, the default input is "normal" and the assumption is that the null statistics follow a F-distribution. See `?lrt` for additional arguments.

5.5.2 Optimal discovery procedure

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

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

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

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

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

5.6 Significance results

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

```
summary(de_odp)
```

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

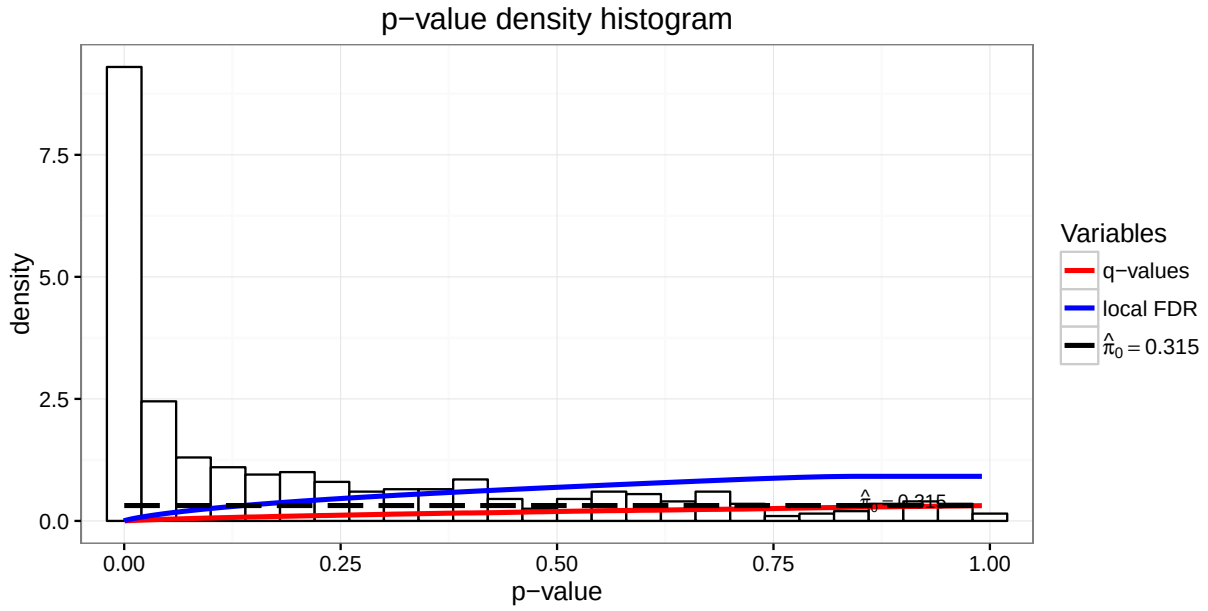

Figure 4: Applying the function `hist` to the slot `qvalueObj` in the `gibson` data set. Function is derived from the `qvalue` package.

```
##
## Biological variables:
##  Null Model:~gender + batch
## <environment: 0x7fb5c4cd8000>
##
##  Full Model:~gender + batch + grp
## <environment: 0x7fb5c4cd8000>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.3154398
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value       61     99   156    191   224  261
## q-value        0     89   156    202   250  313
## local fdr      0     61   106    125   161  196
##            <1
## p-value   500
## q-value   500
## local fdr 500
```

There are three core summaries: `ExpressionSet` summary, `edge` analysis and statistical significance summary. The `ExpressionSet` summary shows a summary of the `ExpressionSet` object. `edge` analysis shows

an overview of the models used and other information about the data set. The significance analysis shows the proportion of null genes, π_0, and significant genes at various cutoffs in terms of p-values, q-values and local false discovery rates.

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

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

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

```
names(sig_results)

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

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

```
hist(sig_results)
```

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

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

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

```
qvalues[1]

## [1] 0.01710459
```

So for this particular gene, the q-value is 0.0171046. If we consider a false discovery rate cutoff of 0.1 then this gene is significant. Therefore, the observed differences observed in Figure **??** are significant so this particular gene is differentially expressed between locations.

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

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

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

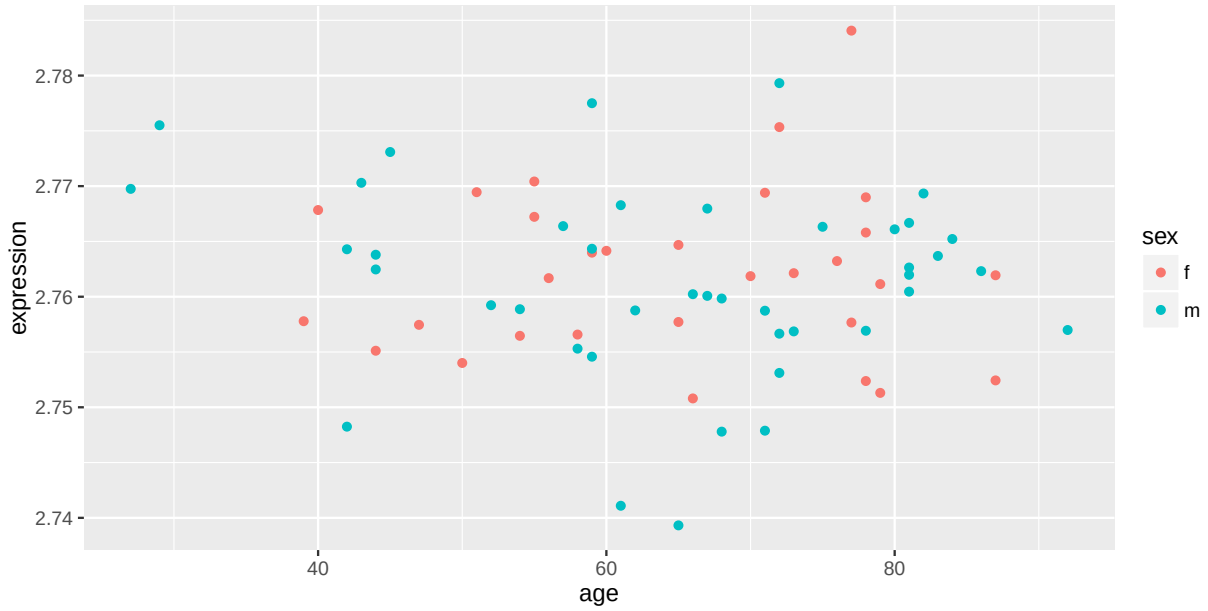

Figure 5: Plot of gene 5 in the `kidney` study.

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 **?**. 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 **?** for additional information regarding the data set.

6.1 Importing the data

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

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

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

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

6.2 Creating the full and null models

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

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

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

```
fullModel(de_obj)

## ~adj.var + ns(tme, df = 4, intercept = FALSE)
## <environment: 0x7fb5c88724d8>

nullModel(de_obj)

## ~adj.var
## <environment: 0x7fb5c88724d8>
```

Notice that the difference between the full and null model is the natural spline fit of the `age` variable. If we look at Figure **??**, 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 **?** for a detailed discussion on modelling in time course studies.

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

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

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

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

6.3 The deSet object

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

```
slotNames(de_obj)
```

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

A description of each slot is listed below:

- `full.model`: the full model of the experiment. Contains the biological variables of interest and the adjustment variables.
- `null.model`: the null model of the experiment. Contains the adjustment variables in the experiment.
- `full.matrix`: the full model in matrix form.
- `null.matrix`: the null model in matrix form.
- `individual`: variable that keeps track of individuals (if same individuals are sampled multiple times).
- `qvalueObj`: `qvalue` list. Contains p-values, q-values and local false discovery rates of the significance analysis. See the `qvalue` package for more details.
- `ExpressionSet`: inherits the slots from `ExpressionSet` object.

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

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

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

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

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

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

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

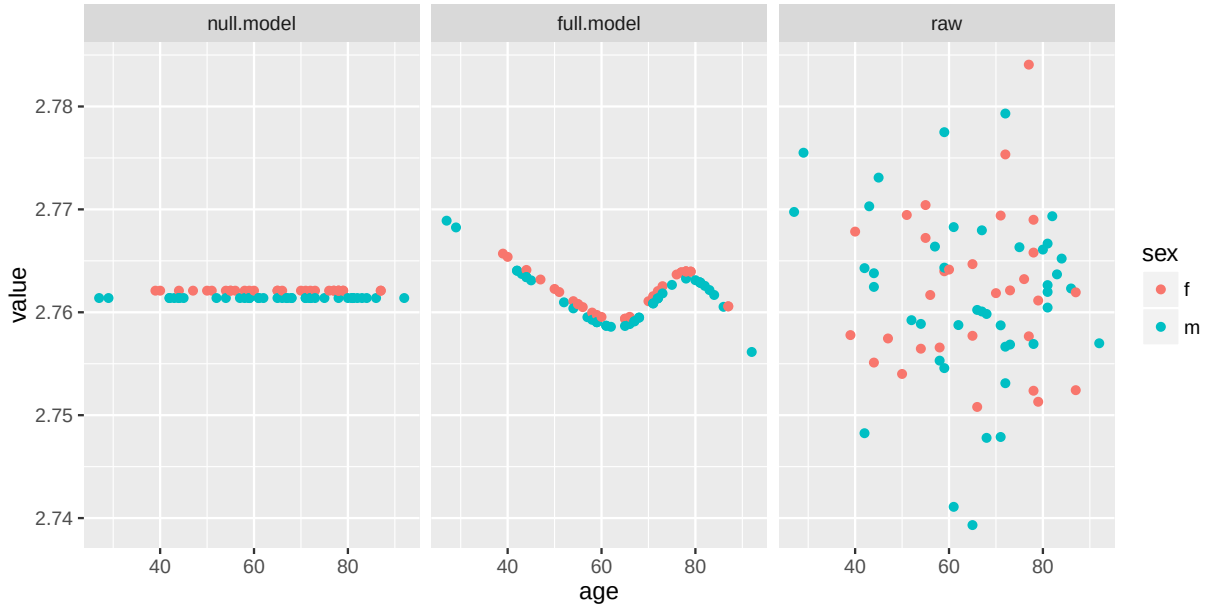

Figure 6: Plot of gene 5 in the `kidney` study after applying the full and null model fit. The "raw" column are the expression values of the original data.

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

6.4 Fitting the data

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

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

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

- `fit.full`: fitted values from the full model.
- `fit.null`: fitted values from null model.
- `res.full`: residuals from the full model.
- `res.null`: residuals from the null model.
- `dH.full`: diagonal elements in the projection matrix for the full model.
- `beta.coef`: the coefficients for the full model.
- `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 **??**. 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 **??**, it seems that the full model fits the data better than the null model. In order to conclude it is significant, we need to calculate the p-value. The user can use either the optimal discovery procedure or likelihood ratio test.

6.5.1 Likelihood ratio test

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

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

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

6.5.2 Optimal discovery procedure

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

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

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

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

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

6.6 Significance results

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

```
summary(de_odp)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 72 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: adj.var tme
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## de Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 500
##
## Biological variables:
##  Null Model:~adj.var
## <environment: 0x7fb5c81bb4e0>
##
##  Full Model:~adj.var + ns(tme, df = 4, intercept = FALSE)
## <environment: 0x7fb5c81bb4e0>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.4452372
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
```

```
## p-value        2      8    32     49    92  145
## q-value        0      0     2      8    26   42
## local fdr      0      0     2      6    10   28
##            <1
## p-value   500
## q-value   500
## local fdr 500
```

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

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

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

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

```
names(sig_results)

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

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

```
hist(sig_results)
```

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

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

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

```
qvalues[5]

## [1] 0.1756436
```

So for this particular gene, the q-value is 0.1756436. If we consider a false discovery rate cutoff of 0.1 then this gene is not significant. Therefore, the observed differences observed in Figure **??** are not significant so

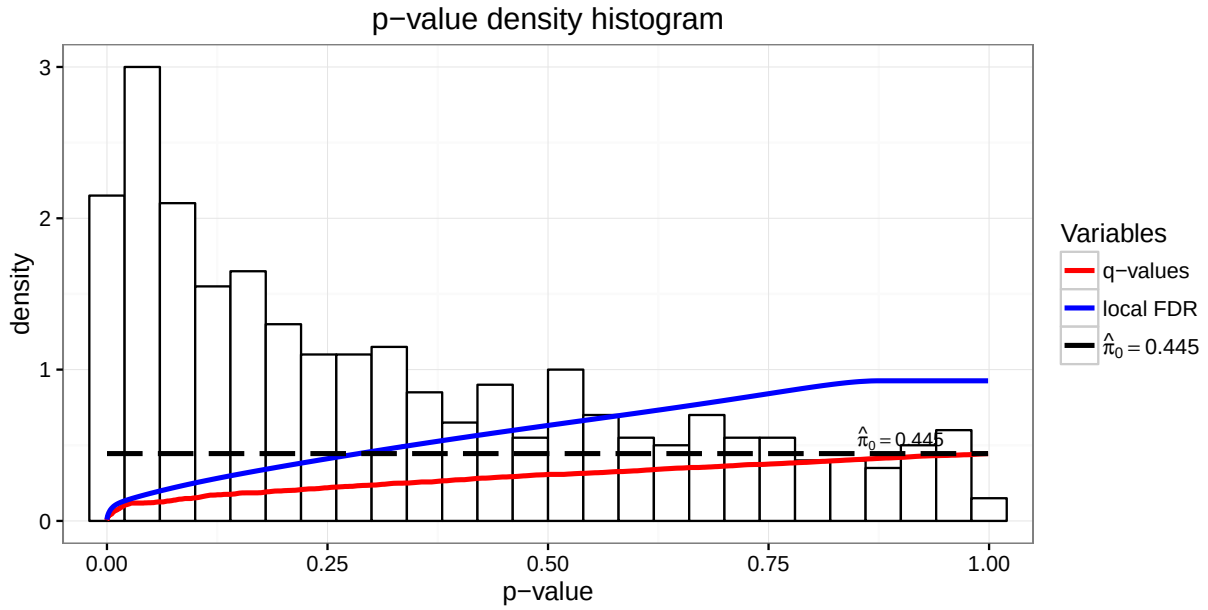

Figure 7: Applying the function `hist` to the slot `qvalueObj` in the `kidney` data set. Function is derived from the `qvalue` package.

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:

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

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

7 Case study: longitudinal time course experiment

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

7.1 Importing the data

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

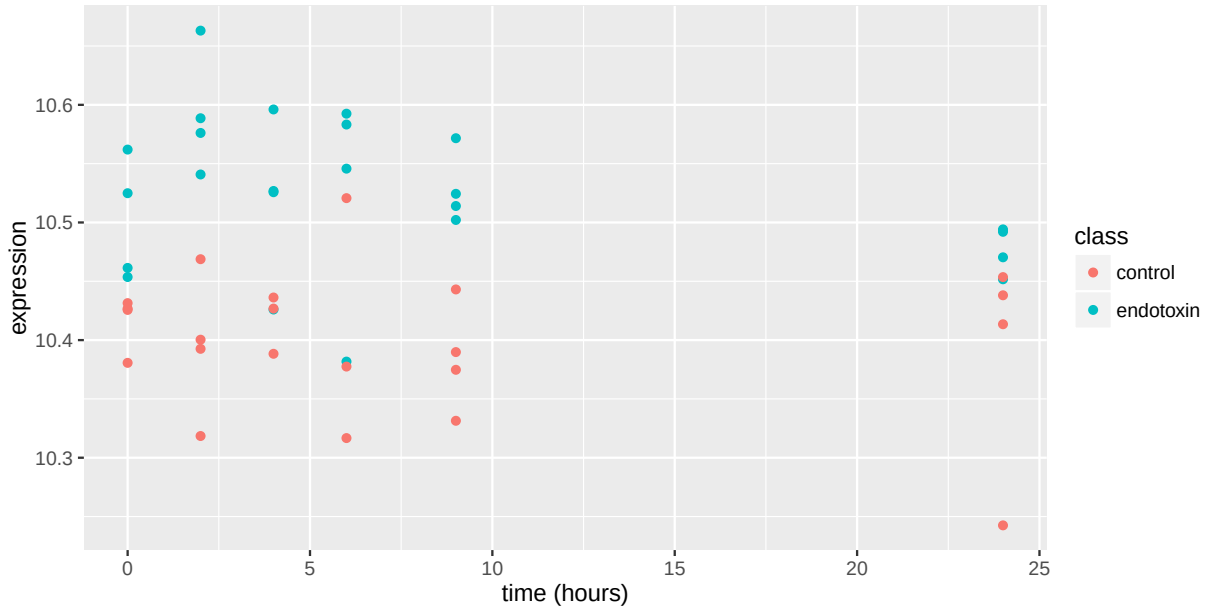

Figure 8: Plot of gene 2 in the `endotoxin` study.

```
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 **??**. It is very difficult to find a trend for this particular gene. Instead, we need to adjust the data with the models in the study.

7.2 Creating the full and null models

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

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

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

variable, `ind` is the `individuals` variable and the `grp` variable is `class`. To view the models created by `build_study`:

```
fullModel(de_obj)

## ~grp + ns(tme, df = 2, intercept = FALSE) + (grp):ns(tme, df = 2,
##     intercept = FALSE)
## <environment: 0x7fb5c4cce230>

nullModel(de_obj)

## ~grp + ns(tme, df = 2, intercept = FALSE)
## <environment: 0x7fb5c4cce230>
```

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

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

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

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

7.3 The `deSet` object

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

```
slotNames(de_obj)

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

A description of each slot is listed below:

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

adjustment variables.

- `null.model`: the null model of the experiment. Contains the adjustment variables in the experiment.
- `full.matrix`: the full model in matrix form.
- `null.matrix`: the null model in matrix form.
- `individual`: variable that keeps track of individuals (if same individuals are sampled multiple times).
- `qvalueObj`: `qvalue` list. Contains p-values, q-values and local false discovery rates of the significance analysis. See the `qvalue` package for more details.
- `ExpressionSet`: inherits the slots from `ExpressionSet` object.

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

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

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

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

```
fullModel(de_obj)

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

nullModel(de_obj)

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

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

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

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

7.4 Fitting the data

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

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

The `stat.type` argument specifies whether you want the `odp` or `lrt` fitted values. The difference between

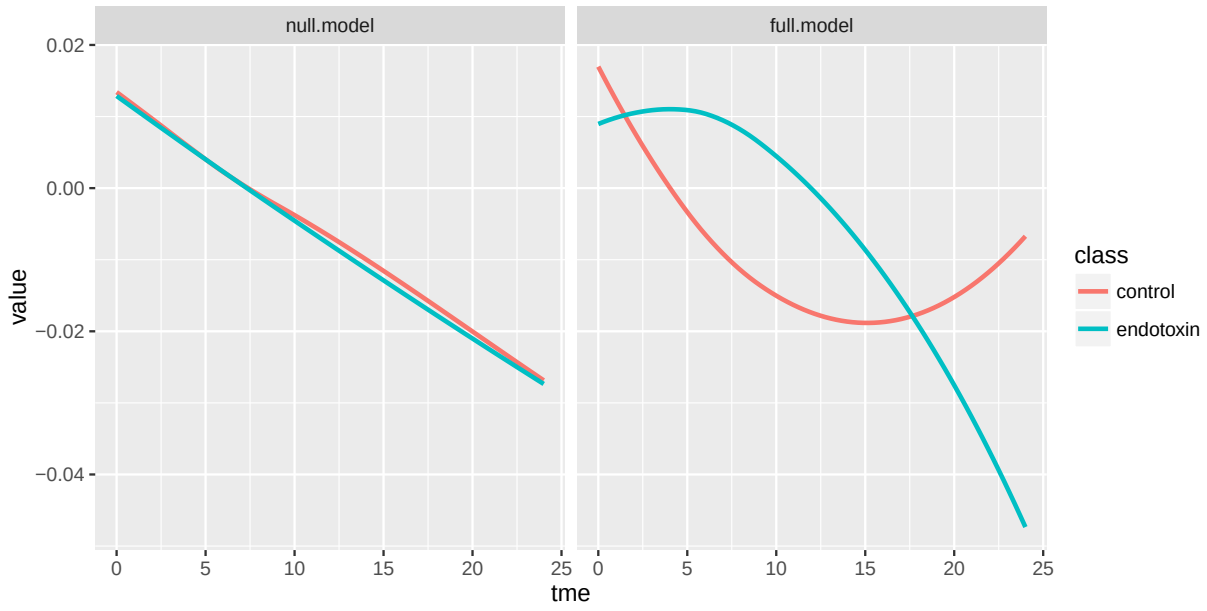

Figure 9: Plot of gene 2 in the `endotoxin` study after applying the full and null model fit. The "raw" column is the expression values of the original data.

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 an `deFit` object. The fitted values of the second gene are shown in Figure **??**. 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 **??**, it seems that the full model fits the data better than the null model. In order to conclude it is significant, we need to calculate the p-value. The user can use either the optimal discovery procedure or likelihood ratio test.

7.5.1 Likelihood ratio test

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

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

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

7.5.2 Optimal discovery procedure

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

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

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

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

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

7.6 Significance results

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

```
summary(de_odp)

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 46 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 46 (46 total)
##   varLabels: tme grp
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## de Analysis Summary
##
## Total number of arrays: 46
## Total number of probes: 500
##
## Biological variables:
##  Null Model:~grp + ns(tme, df = 2, intercept = FALSE)
## <environment: 0x7fb5badb0ea8>
##
##  Full Model:~grp + ns(tme, df = 2, intercept = FALSE) + (grp):ns(tme, df = 2,
##     intercept = FALSE)
## <environment: 0x7fb5badb0ea8>
##
## 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: 0.3518397
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value       15     25    63     90   116  155
## q-value        0     15    38     62    92  141
## local fdr      0      0    21     38    51   84
```

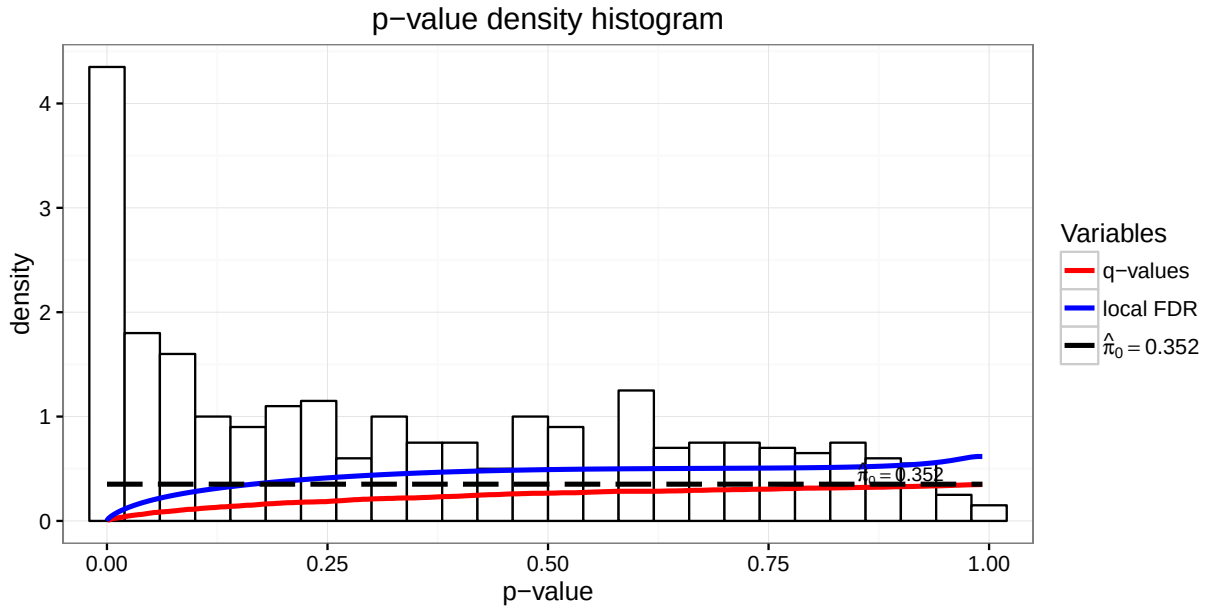

Figure 10: Applying the function `hist` to the slot `qvalueObj` in the `endotoxin` data set. Function is derived from the `qvalue` package.

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

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

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

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

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

```
names(sig_results)

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

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

```
hist(sig_results)
```

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

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

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

```
qvalues[2]

## [1] 0.2528448
```

So for this particular gene, the q-value is 0.2528448. If we consider a false discovery rate cutoff of 0.1 then this gene is not significant. Therefore, the observed differences observed in Figure **??** are not significant so this particular gene is not differentially expressed between `class` as time varies.

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

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

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

8 sva: Surrogate variable analysis

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

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

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

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

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

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

```
fullModel(de_sva)

## ~SV1 + SV2 + SV3 + sex + ns(age, df = 4)
## <environment: 0x7fb5ab0655b8>

nullModel(de_sva)

## ~SV1 + SV2 + SV3 + sex
## <environment: 0x7fb5ab0655b8>
```

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

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

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

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

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

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

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 72 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: sex age ... SV3 (5 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## de Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 500
```

```
##
## Biological variables:
##  Null Model:~SV1 + SV2 + SV3 + sex
## <environment: 0x7fb5ab0655b8>
##
##  Full Model:~SV1 + SV2 + SV3 + sex + ns(age, df = 4)
## <environment: 0x7fb5ab0655b8>
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.3267956
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value        2      9    31     48    79  125
## q-value        0      0     2     11    30   79
## local fdr      0      0     2      7    20   36
##            <1
## p-value   500
## q-value   500
## local fdr 500
```

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

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

9 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 **?**. The `snm` package is implemented in the `apply_snm` function. Continuing the analysis on the `kidney` study in **??**:

```
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:

```
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`:

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

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

```
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: 0.3027327
##
## Cumulative number of significant calls:
##
```

```
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value        1      9    37     68   104  160
## q-value        0      0     1     17    63  180
## local fdr      0      0     1     15    27   80
##            <1
## p-value   500
## q-value   500
## local fdr 500
```

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

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

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

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

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

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

```
##   old_pi0est new_pi0est
## 1  0.4085672  0.4457143
```

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, lets create an `ExpressionSet` object from the `kidney` dataset:

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

In the `kidney` experiment they were interested in finding the effect of age on gene expression. In this case, we handle the time variable, `age`, by fitting a natural spline curve as done in **?**. The relevant models for the experiment can be written as

```
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:

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

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

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

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

##
## ExpressionSet Summary
##
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 72 samples
##   element names: exprs
```

```
## protocolData: none
## phenoData
##   sampleNames: 1 2 ... 72 (72 total)
##   varLabels: age sex
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
##
## de Analysis Summary
##
## Total number of arrays: 72
## Total number of probes: 500
##
## Biological variables:
##  Null Model:~1 + sex
##
##  Full Model:~1 + sex + ns(age, intercept = FALSE, df = 4)
##
## .......
##
##
## Statistical significance summary:
## pi0: 0.3727618
##
## Cumulative number of significant calls:
##
##           <1e-04 <0.001 <0.01 <0.025 <0.05 <0.1
## p-value        2      4    29     59    89  143
## q-value        0      0     3     11    27   87
## local fdr      0      0     2      4    18   43
##            <1
## p-value   500
## q-value   500
## local fdr 500
```

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

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

Acknowledgements

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