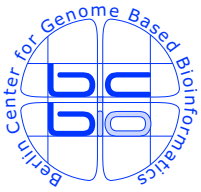

Max Planck Institute for Molecular Genetics
Computational Diagnostics Group @ Dept. Vingron
Ihnestrasse 63-73, D-14195 Berlin, Germany
http://compdiag.molgen.mpg.de/

Estimation of local false discovery rate

User's guide to the Bioconductor package *twilight* 1.0.2

Stefanie Scheid[1] **and Rainer Spang**

Max Planck Institute for Molecular Genetics
Computational Diagnostics
Department for Computation Molecular Biology
Ihnestr. 63-73, D-14195 Berlin, Germany

Technical Report
Nr. 2004/01

Abstract

This is the vignette of the Bioconductor compliant package *twilight*. We describe our implementation of a stochastic search algorithm to estimate the local false discovery rate. In addition, the package provides functions to test for differential gene expression in the common two-condition setting.

[1]Corresponding author: `stefanie.scheid@molgen.mpg.de`

Contents

Chapter 1

Introduction

In a typical microarray setting with gene expression data observed under two conditions, the local false discovery rate describes the probability that a gene is not differentially expressed between the two conditions given its corrresponding observed score or p-value level. The resulting curve of p-values versus local false discovery rate offers an insight into the **twilight zone** between clear differential and clear non-differential gene expression. The Bioconductor compliant package *twilight* contains two main functions: Function `twilight.pval` performs a two-condition test on differences in means for a given input matrix or expression set (*exprSet*) and computes permutation based p-values. Function `twilight` performs the successive exclusion procedure described in Scheid and Spang (2004) [4] to estimate local false discovery rates and effect size distributions.

Chapter 2

Implemented methods

2.1 Function `twilight.pval`

```
twilight.pval(xin, yin, method="fc", paired=FALSE, B=10000,
yperm=NULL, balance=FALSE, quant.ci=0.95, s0=NULL, verbose=TRUE)
```

The input object `xin` is either a pre-processed gene expression set of class *exprSet* or any data matrix where rows correspond to genes and columns to samples. Each sample was taken under one of two distinct conditions, for example under treatment A or treatment B. The functions in package *twilight* are not limited to microarray data only but can be applied to any two-sample data matrix. However, it is necessary for both expression set or numerical matrix that values are on **additive scale** like log or arsinh scale. The function does not check or transform the data to additive scale. The input vector `yin` contains condition labels of the samples. Vector `yin` has to be numeric and dichotomous. Note that in terms of *under-* and *over*-expression, the samples of the higher labeled condition are compared to the samples of the lower labeled condition.

We are given a pre-processed matrix for samples belonging to two distinct conditions A and B, and gene expression values on additive scale. For gene i in the experiment ($i = 1, \ldots, N$), $\bar{\alpha}_i$ is the mean expression under condition A and $\bar{\beta}_i$ is the mean expression under condition B. To test the null hypothesis of no differential gene expression, function `twilight.pval` compares the mean expression levels $\bar{\alpha}_i$ and $\bar{\beta}_i$. The current version offers three test variants: The classical t-test uses score T_i with

$$T_i = \frac{\bar{\alpha}_i - \bar{\beta}_i}{s_i}, \tag{2.1}$$

where s_i denotes the root pooled variance. The t-test is called with `method="t"`.

The t-test score can be misleadingly high if s_i is very small. To overcome this problem, the Z-test enlarges the denominator by a fudge factor s_0 [6], [1]:

$$Z_i = \frac{\bar{\alpha}_i - \bar{\beta}_i}{s_i + s_0}. \tag{2.2}$$

The Z-test is called with `method="z"`. Fudge factor s_0 is set to `s0=NULL` by default and is only evaluated if `method="z"`. In that case, it is the median of all root pooled variances $s_1, \ldots, s_N$. However, the fudge factor can be chosen manually. Note that if `method="z"` is chosen with `s0=0`, the test call is altered to `method="t"`, the t-test as described above.

The third variant is based on log ratios only with score

$$F_i = \bar{\alpha}_i - \bar{\beta}_i. \tag{2.3}$$

The distribution of scores F_i under the alternative is called *effect size distribution*. With expression values on log or arsinh scale, $\exp(|F_i|)$ is an estimator for the fold change. We call $\exp(|F_i|)$ the *fold change equivalent score* [4]. Note that the package contains a function for plotting the effect size distribution which is only available if function `twilight.pval` was run with `method="fc"`, the fold change test.

Function `twilight.pval` handles paired and unpaired data. In the unpaired case (`paired=FALSE`), only one microarray was hybridized for each patient, like in a treatment and control group setting. In the paired case (`paired=TRUE`), we observed expression values of the same patient under both conditions. The typical example are before and after treatment experiments, where each patient's expression was measured twice. The input arguments `xin` and `yin` do not need to be ordered in a specific manner. It is only necessary that samples within each group have the same order, such that the first samples of the two groups represent the first pair and so on.As an example, we apply function `twilight.pval` on the leukemia data set of Golub et al. (1999) [3] as given in `library(golubEsets)`. For normalization, apply the variance-stabilizing method `vsn` in `library(vsn)` [2].

```
> data(golubMerge)
> golubNorm <- vsn(exprs(golubMerge))
> id <- as.numeric(golubMerge$ALL.AML)
```

There are 72 samples either expressing acute lymphoblastic leukemia (ALL) or acute myeloid leukemia (AML). As the AML patients are labeled with "2" and ALL with "1", we compare AML to ALL expression.

```
> golubMerge$ALL.AML

 [1] ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
[17] ALL ALL ALL ALL AML AML AML AML AML AML AML AML AML AML AML AML AML
[33] AML AML ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
[49] ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL AML AML AML
[65] AML AML AML AML AML AML AML AML
Levels: ALL AML

> id

 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
[34] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
[67] 2 2 2 2 2 2
```

Additionally to computation of scores, empirical p-values are calculated. Argument `B` specifies the number of permutations with default set to `B=10000`. The distribution of scores under the null hypothesis is estimated by computing test scores from the same input matrix with randomly permuted class labels. These permutations are either balanced or unbalanced, with default `balance=FALSE`. The permutation options are described in detail

in section 2.3. For computing empirical p-values, we count for each gene how many of its absolute permutation scores exceed the absolute observed score, and divide by `B`.

Permutation scores are also used to compute expected scores as described in Tusher et al. (2001) [6]. In addition, we compute confidence bounds for the maximum absolute difference of each set of permutation scores to expected scores. The width of the confidence bound is chosen with `quant.ci`. With default `quant.ci=0.95`, the maximum absolute difference of permutation to expected scores exceeded the confidence bound in only 5% of all permutations.

Using the optional argument `yperm`, a user-specified permutation matrix can be passed to the function. In that case, `yperm` has to be a *binary* matrix where each row is one vector of permuted class labels. The label "1" in `yperm` corresponds to the higher labeled original class. If the permutation matrix is specified, no other permutation is done and argument `B` will be ignored. Besides `set.seed`, argument `yperm` can be used to reproduce results by fixing the matrix of random permutations.

Continuing the example above, we perform a fold change test on the expression data in `golubNorm` which was transformed to arsinh scale by normalization with `vsn`. As an illustration, we apply function `twilight.pval` with 1000 permutations only.

```
> library(twilight)
> pval <- twilight.pval(golubNorm, id, B = 1000)

No complete enumeration. Prepare permutation matrix.
Compute vector of observed statistics.
Compute expected scores and p-values.
Compute q-values.
Compute values for confidence lines.
```

The function checks whether complete enumeration of all permutations is possible. Complete enumeration is performed as long as the number of permutations does not exceed 10 000. The admissible sample sizes depend on whether the experiment was paired or unpaired and whether balanced or unbalanced permutations are used. Details are given in section 2.3. The values in the accompanying data set `expval` were computed in the same manner but with 10 000 permutations.

```
> data(expval)
> expval$call

[1] "Test: fc. Paired: FALSE. Number of permutations: 10000. Balanced: FALSE."
```

The output object of function `twilight.pval` is of class *twilight* with several elements stored in a list.

```
> class(expval)

[1] "twilight"

> names(expval)

[1] "result"   "ci.line"  "quant.ci" "lambda"   "pi0"      "boot.pi0"
[7] "boot.ci"  "effect"   "call"
```

The element `quant.ci` contains the corresponding input value which is passed to the plotting function. Element `ci.line` is used for plotting confidence bounds and contains the computed quantile of maximum absolute differences. The output dataframe `result` contains a matrix with several columns.

```
> names(expval$result)
```

```
[1] "observed"  "expected"  "candidate" "pvalue"    "qvalue"
[6] "fdr"       "mean.fdr"  "lower.fdr" "upper.fdr"
```

The dataframe stores observed and expected scores and corresponding empirical p-values. The genes are ordered by p-value and transcript name. Genes with observed score exceeding the confidence bounds are marked as "1" in the binary vector `result$candidate`. The output object is passed to function `plot.twilight` to produce a plot as in Tusher et al. (2001) [6] with additional confidence lines and genes marked as candidates, see Figure 2.1.

```
> expval$result[1:10, 1:5]
```

```
              observed   expected candidate pvalue      qvalue
AB000449_at -0.5591418 -0.2466945         0  1e-04 0.001851991
AB002559_at  0.4712372  0.2355924         0  1e-04 0.001851991
AF005043_at -0.3133348 -0.1397644         0  1e-04 0.001851991
AF009426_at -0.4699483 -0.2045175         0  1e-04 0.001851991
D00763_at   -0.5043169 -0.2210751         0  1e-04 0.001851991
D10495_at    1.5156898  0.5192884         1  1e-04 0.001851991
D13627_at   -0.7540288 -0.3302692         0  1e-04 0.001851991
D14658_at   -0.6748078 -0.3026529         0  1e-04 0.001851991
D14664_at    0.8451562  0.3539611         0  1e-04 0.001851991
D21262_at   -0.7137899 -0.3142349         0  1e-04 0.001851991
```

Note that empirical p-values are not a monotone transformation of the test scores. For each gene separately, its observed score is compared to its scores under permutation. Consider two genes with equal observed score. The permutation score distributions of the two genes might be different. Thus, the resulting p-values differ although the scores are equal. The concept of comparison of observed and expected scores is completely different. The comparison is not done on a gene-wise level but by taking all genes at once into account. Therefore, a candidate gene might have a high difference between observed and expected score but a rather high p-value. We recommend to base inference on gene-wise empirical p-values, not on differences.

In addition, q-values and the estimated percentage of non-induced genes π_0 are computed as described in Remark B of Storey and Tibshirani (2003) [5]. These are stored in `result$qvalue` (see above) and `pi0`. The remaing output elements of `expval` are left free to be filled by function `twilight`. With `"qvalues"`, Figure 2.2 shows the plot of q-values against the corresponding number of rejected hypotheses.

```
> expval$pi0
```

```
[1] 0.5799648
```

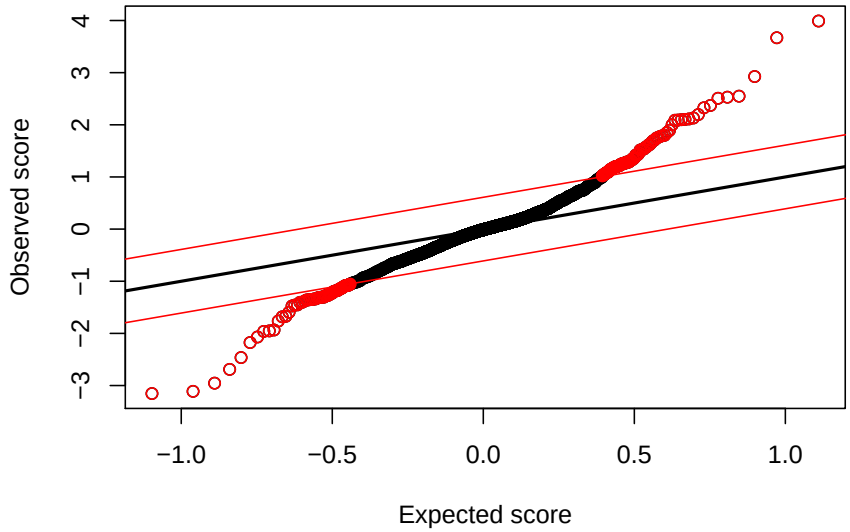

Figure 2.1: Expected versus observed test scores. Deviation from the diagonal line gives evidence for differential expresion. The red lines mark the 95% confidence interval on the absolute difference between oberved and expected scores. The plotting call is `plot(expval,which="scores",grayscale=F,legend=T)`.

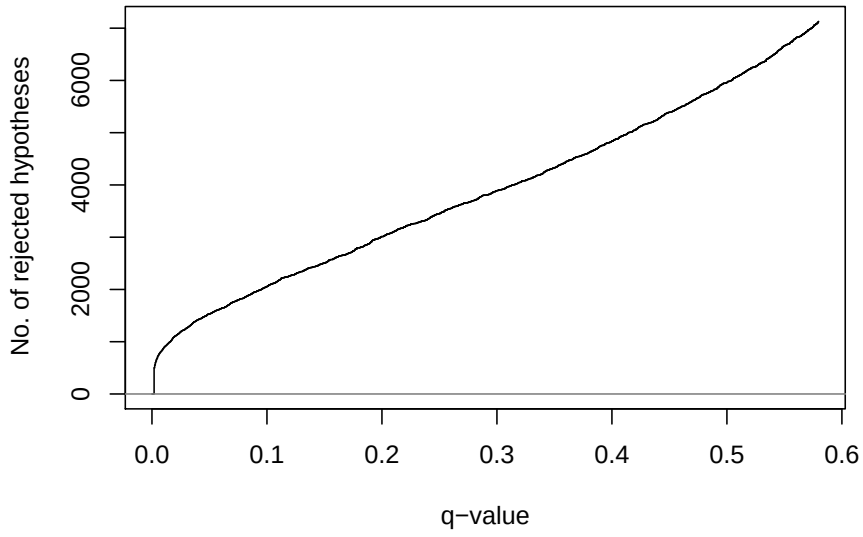

Figure 2.2: Stairplot of q-values against the resulting size of list of significant genes. A list containing all genes with $q \leq q_0$ has an estimated global false discovery rate of q_0. The plotting call is `plot(expval,which="qvalues")`.

2.2 Function `twilight`

```
twilight(xin, lambda=NULL, B=0, boot.ci=0.95, clus=NULL, verbose=TRUE)
```

Local false discovery rates (fdr) are estimated from a simple mixture model given the density $f(t)$ of observed scores $T = t$:

$$f(t) = \pi_0 \, f_0(t) + (1 - \pi_0) \, f_1(t) \quad \Rightarrow \quad \text{fdr}(t) = \pi_0 \, \frac{f_0(t)}{f(t)}, \tag{2.4}$$

where $\pi_0 \in [0, 1]$ is the overall percentage of non-induced genes. Terms f_0 and f_1 are score densities under no induction and under induction respectively. Assume that there exists a transformation W such that $U = W(T)$ is uniformly distributed in $[0, 1]$ for all genes not differentially expressed. In a multiple testing scenario these u-values are p-values corresponding to the set of observed scores. However, we do not regard the local false discovery rate as a multiple error rate but as an exploratory tool to describe a microarray experiment over the whole range of significance, thus speaking of u-values instead of p-values.

Mapping scores to u-values allows to assume $f_0(u)$ to be the uniform density instead of specifying the null density $f_0(t)$ with respect to a chosen scoring method. The implemented successive exclusion procedure (SEP) splits any vector of u-values into a uniformly distributed null part and an alternative part. The uniform part represents genes that are not differentially expressed. Based on histogram counts of the null and the alternative part, we estimate local false discovery rates. These denote the conditional probability of being non-induced given we observe a certain u-value level. The proportion of the uniform part to the total number of genes in the experiment is a natural estimator for percentage π_0. The successive exclusion procedure is described in detail in Scheid and Spang (2004) [4]. The functionality of `twilight` is not limited to microarray experiments. In principle, any vector of u-values can be passed to `twilight` as long as the assumption of uniformity under the null hypothesis is valid.

The objective function in `twilight` includes a penalty term that is controlled by regularization parameter $\lambda \geq 0$. The regularization ensures that we find a separation such that the uniform part contains as many u-values as possible. As percentage π_0 is often underestimated, the inclusion of a penalty term results in a more "conservative" estimate that is usually less biased. If not specified (`lambda=NULL`), function `twilight.getlambda` finds a suitable λ.

The estimates for probability π_0 and local false discovery rate are averaged over 10 runs of SEP. In addition, bootstrapping can be performed to give bootstrap estimates and bootstrap percentile confidence intervals on both π_0 and local false discovery rate. The number of bootstrap samples is set by argument `B`, and the width of the bootstrap confidence interval is set by argument `boot.ci`.

Function `twilight` takes *twilight* objects or any vector of u-values as input and returns a *twilight* object. If the input is of class *twilight*, the function works on the set of empirical p-values and fills in the remaining output elements. Note that the estimate for π_0 is replaced, and q-values are recalculated with the new estimate π_0. As an example, we run SEP with 1000 bootstrap samples and 95% boostrap confidence intervals: `twilight(xin=expval, B=1000, boot.ci=0.95)`, as was done for data set `exfdr`.

```
> data(exfdr)
> exfdr$lambda

[1] 0.025

> exfdr$pi0

[1] 0.5914855

> exfdr$boot.pi0

       pi0 lower.pi0 upper.pi0
 0.5914281  0.563894 0.6207112

> exfdr$result[1:5, 6:9]

                   fdr   mean.fdr  lower.fdr  upper.fdr
AB000449_at 0.04504678 0.05547103 0.04409335 0.07451714
AB002559_at 0.04504678 0.05547103 0.04409335 0.07451714
AF005043_at 0.04504678 0.05547103 0.04409335 0.07451714
AF009426_at 0.04504678 0.05547103 0.04409335 0.07451714
D00763_at   0.04504678 0.05547103 0.04409335 0.07451714
```

The output elements `result$fdr`, `result$mean.fdr`, `result$lower.fdr` and `result$upper.fdr` contain the estimated local false discovery rate, the bootstrap average and upper and lower bootstrap confidence bounds. These values are used to produce the following plots which are only available after application of function `twilight`. First, we plot u-values against the corresponding conditional probabilities of being induced given the u-value level, that is $1-\text{fdr}$, see Figure 2.3. Going back to observed scores, we produce a *volcano plot*, that is observed scores versus local false discovery rate, see Figure 2.4.

Output element `effect` contains histogram information about the effect size distribution, that is log ratio under the alternative. One run of the successive exclusion procedure results in a split of the input u-value vector into a null and an alternative part. We estimate the effect size distribution from the distribution of log ratio scores corresponding to u-values in the alternative part. Again, this estimate is averaged over 10 runs of the procedure. Argument `which="effectsize"` produces the histogram of all observed log ratios overlaid with the averaged histogram of log ratios in the alternative, see Figure 2.5. The x-axis is changed to fold change equivalent scores or rather to increase in effect size. Given an observed log ratio F, the increase in effect size is $(\exp(|F|)-1)\cdot\text{sign}(F)\cdot 100\%$. A value of 0% corresponds to no change (fold change of 1), a value of 50% to fold change 1.5 and so on. A value of -100% corresponds to a 2-fold down-regulation, that is fold change of 0.5.

The last plotting argument `which="table"` tabulates the histogram information in terms of fold change equivalent scores and log ratios.

```
> tab <- plot(exfdr, which = "table")
> tab[1:8, ]
```

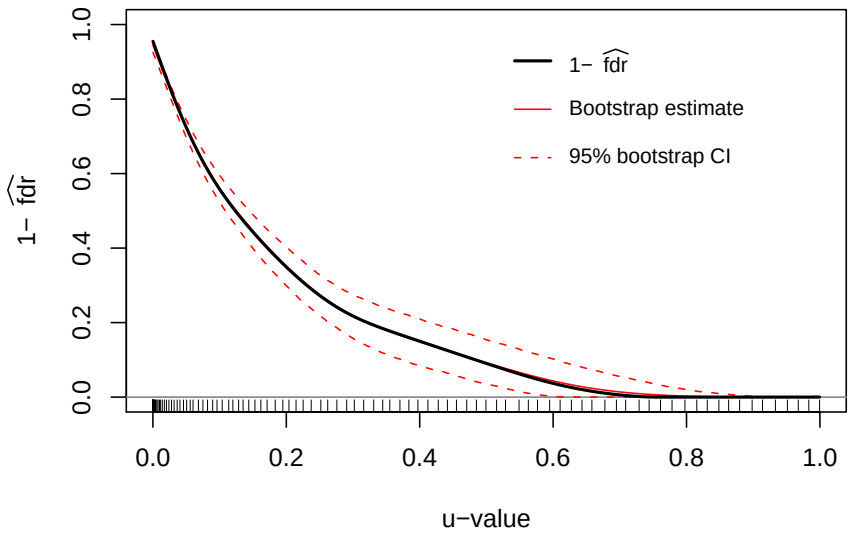

Figure 2.3: Curve of estimated local false discovery over u-values. The red lines denote the bootstrap mean (solid line, virtually not visible) and the 95% bootstrap confidence interval on the local false discovery rate (dashed lines). The bottom ticks are 1% quantiles of u-values. The plotting call is `plot(exfdr,which="fdr",grayscale=F,legend=T)`.

	LogRatio	Mixture	Alternative
-2234%	-3.15	2	1.9
-2012%	-3.05	0	0.0
-1811%	-2.95	1	1.0
-1629%	-2.85	0	0.0
-1464%	-2.75	0	0.0
-1315%	-2.65	1	0.9
-1181%	-2.55	0	0.0
-1059%	-2.45	1	1.0

The input argument `clus` of function `twilight` is used to perform parallel computation within `twilight`. Parallelizing saves computation time which is especially useful if the number of bootstrap samples `B` is large. With default `clus=NULL`, no parallelizing is done. If specified, `clus` is passed as input argument to `makeCluster` in `library(snow)`. Please make sure that `makeCluster(clus)` works properly in your environment.

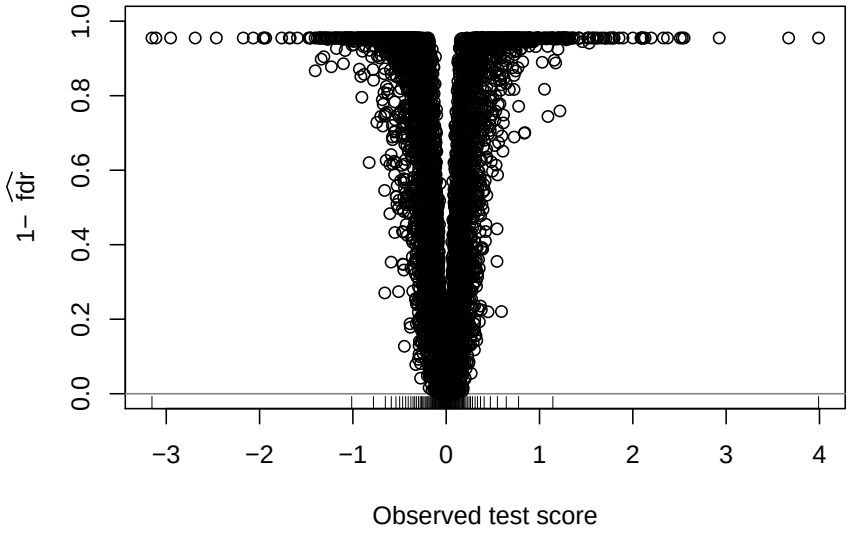

Figure 2.4: Volcano plot of observed test scores versus local false discovery rate. The bottom ticks are 1% quantiles of observed scores. The plotting call is `plot(exfdr,which="volcano")`.

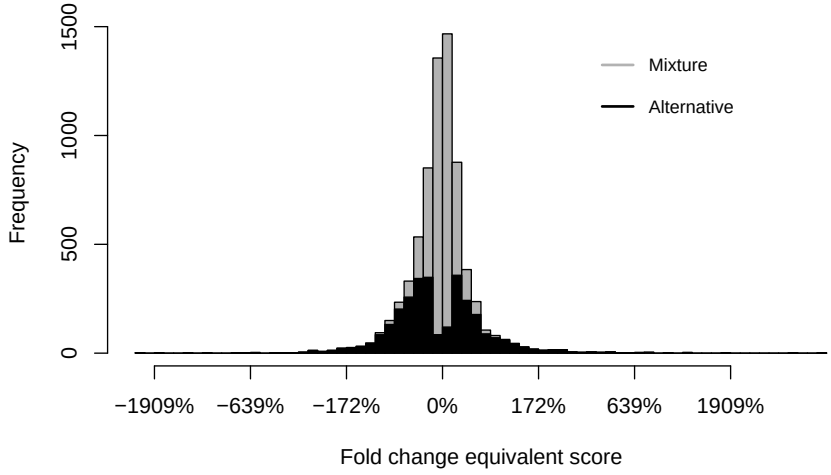

Figure 2.5: Observed effect size distribution (gray histogram) overlaid with the estimated effect size distribution under the null hypothesis (black histogram). The plotting call is `plot(exfdr,which="effectsize",legend=T)`.

2.3 Function `twilight.combi`

```
twilight.combi(xin, pin, bin)
```

Function `twilight.combi` is used within `twilight.pval` to completely enumerate all permutations of a *binary* input vector `xin`. Argument `pin` specifies whether the input vector corresponds to paired or unpaired data. Argument `bin` specifies whether permutations are balanced or unbalanced. Note that the resulting permutations are always "as balanced as possible": The balancing is done for the smaller subsample. If its sample size is odd, say 7, `twilight.combi` computes all permutations with 3 and 4 samples unchanged.

As first example, compute all unbalanced permutations of an unpaired binary vector of length 5 with two zeros and three ones. The number of rows are

$$m = \frac{5!}{2! \cdot 3!} = 10. \tag{2.5}$$

```
> x <- c(rep(0, 2), rep(1, 3))
> x

[1] 0 0 1 1 1

> twilight.combi(x, pin = FALSE, bin = FALSE)

      [,1] [,2] [,3] [,4] [,5]
 [1,]    0    0    1    1    1
 [2,]    0    1    0    1    1
 [3,]    0    1    1    0    1
 [4,]    0    1    1    1    0
 [5,]    1    0    0    1    1
 [6,]    1    0    1    0    1
 [7,]    1    0    1    1    0
 [8,]    1    1    0    0    1
 [9,]    1    1    0    1    0
[10,]    1    1    1    0    0
```

Each row contains one permutation. The first row contains the input vector. In balanced permutations, we omit the input vector and those rows where both original zeros have been shifted to the last three columns. The number of balanced rows is

$$m = \binom{2}{1} \cdot \frac{3!}{1! \cdot 2!} = 6. \tag{2.6}$$

```
> twilight.combi(x, pin = FALSE, bin = TRUE)

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    0    1    1
[2,]    0    1    1    0    1
[3,]    0    1    1    1    0
[4,]    1    0    0    1    1
[5,]    1    0    1    0    1
[6,]    1    0    1    1    0
```

Next, consider a paired input vector with four pairs. The first zero and the first one are the first pair and so on. In paired settings, values are flipped only within a pair. The number of rows is

$$m = \frac{1}{2} \cdot 2^4 = 2^3 = 8. \tag{2.7}$$

```
> y <- c(rep(0, 4), rep(1, 4))
> y

[1] 0 0 0 0 1 1 1 1

> twilight.combi(y, pin = TRUE, bin = FALSE)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    0    0    0    1    1    1    1
[2,]    0    0    0    1    1    1    1    0
[3,]    0    0    1    0    1    1    0    1
[4,]    0    1    0    0    1    0    1    1
[5,]    1    0    0    0    0    1    1    1
[6,]    0    0    1    1    1    1    0    0
[7,]    0    1    0    1    1    0    1    0
[8,]    0    1    1    0    1    0    0    1
```

The matrix above contains only half of all possible $2^4 = 16$ permutations. The reversed case `1 - twilight.combi(y, pin=TRUE, bin=FALSE)` is omitted as this will lead to the same absolute test scores as `twilight.combi(y, pin=TRUE, bin=FALSE)`. The same concept applies to balanced paired permutations. Now, two pairs are kept fixed and two pairs are flipped in each row. The number of balanced rows is

$$m = \frac{1}{2} \cdot \binom{4}{2} = 3. \tag{2.8}$$

```
> twilight.combi(y, pin = TRUE, bin = TRUE)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    0    1    1    1    1    0    0
[2,]    0    1    0    1    1    0    1    0
[3,]    0    1    1    0    1    0    0    1
```

The complete enumeration of `twilight.combi` is limited by the sample sizes. The function returns `NULL` if the resulting number of rows exceeds 10 000. If `NULL` is returned, function `twilight.pval` uses the functions `twilight.permute.unpair` and `twilight.permute.pair` which return a matrix of random permutations. For example, use the latter function to compute 7 balanced permutations of the paired vector `y`.

```
> twilight.permute.pair(y, 7, bal = TRUE)

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    1    0    0    0    0    1    1
[2,]    0    0    1    1    1    1    0    0
```

```
[3,]    0    0    1    1    1    1    0    0
[4,]    1    1    0    0    0    0    1    1
[5,]    0    0    1    1    1    1    0    0
[6,]    0    1    1    0    1    0    0    1
[7,]    0    0    1    1    1    1    0    0
```

Acknowledgements

This work was done within the context of the Berlin Center for Genome Based Bioinformatics (BCB), part of the German National Genome Network (NGFN), and supported by BMBF grants 031U109C and 03U117 of the German Federal Ministry of Education.

Chapter 3

Bibliography

[1] B. Efron, R. Tibshirani, J.D. Storey and V.G. Tusher, "Empirical Bayes Analysis of a Microarray Experiment", *J. Am. Stat. Assoc.*, vol. 96, no. 456, pp. 1151-1160, 2001.

[2] W. Huber, A. von Heydebreck, H. Sültmann, A. Poustka and M. Vingron, "Variance stabilization applied to microarray data calibration and to the quantification of differential expression", *Bioinformatics*, vol. 18, suppl. 1, pp. S96-S104, 2002.

[3] T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M.L. Loh, J.R. Downing, M.A. Caligiuri, C.D. Bloomfield and E.S. Lander, "Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring", *Science*, vol. 286, pp. 531-537, 1999.

[4] S. Scheid and R. Spang, "A stochastic downhill search algorithm for estimating the local false discovery rate", *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 1, no. 3, pp. 98-108, 2004.

[5] J.D. Storey and R. Tibshirani, "Statistical significance for genomewide studies", *Proc. Natl. Acad. Sci.*, vol. 100, no. 16, pp. 9440-9445, 2003.

[6] V. Tusher, R. Tibshirani and C. Chu, "Significance analysis of microarrays applied to ionizing radiation response", *Proc. Natl. Acad. Sci.*, vol. 98, no. 9, pp. 5116-5121, 2001.