Estimation of Local False Discovery Rates

User's Guide to the Bioconductor Package *twilight* 1.6.1

Stefanie Scheid and Rainer Spang

email: `first.last@molgen.mpg.de`

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.

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. The package is also described in a short application note [5].

Acknowledgements

Chapter 2

Implemented functions

2.1 `twilight.pval`: Testing effect sizes

```
twilight.pval(xin, yin, method="fc", paired=FALSE, B=1000, 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 preprocessed 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 pooled standard deviation. 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 [7], [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 the pooled standard

deviations $s_1, \ldots, s_N$. 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 equivalent 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. However, the order of the samples in `xin` has to equal the order in `yin`.

As an example, we apply function `twilight.pval` on the training set of the leukemia data of Golub et al. (1999) [3] as given in `library(golubEsets)`. For normalization, apply the variance-stabilizing method `vsn` in `library(vsn)` [2].

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

There are 38 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.

```
> golubTrain$ALL.AML

 [1] ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL ALL
[18] ALL ALL ALL ALL ALL ALL ALL ALL ALL AML AML AML AML AML AML AML
[35] 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 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[36] 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=1000`. 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 2.5. For computing empirical *p*-values, we count for each gene how many of *all* absolute permutation scores exceed the absolute observed score, and divide by `B`·(number of genes).

Permutation scores are also used to compute expected scores as described in Tusher et al. (2001) [7]. 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. Please note that the first row of `yperm` must be the input vector `yin`. Otherwise, the *p*-value calculation will be incorrect.

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`. We do a quick example with few permutations.

```
> library(twilight)
> pval <- twilight.pval(golubNorm, id, B = 100)


No complete enumeration. Prepare permutation matrix.
Compute vector of observed statistics.
Compute expected scores and p-values. This will take approx. 14 seconds.
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 the value set by `B`. Thus, if you want to turn off the compulsive enumeration and use all possible permutations, you need to select a small `B` or simply keep the default `B=1000`. Details on the enumeration functions are given in section 2.5. The values in the accompanying data set `expval` were computed in the same manner as in the example above but with the complete data set `data(golubMerge)` in `library(golubEsets)` and 1000 permutations.

```
> data(expval)
> expval
```

```
Twilight object with
    7129 transcripts
    observed and expected test statistics
    p- and q-values

Estimated percentage of non-induced genes:
      pi0
0.6197128


Function call:
Test: fc. Paired: FALSE. Number of permutations: 1000. 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" "index"
```

The dataframe stores observed and expected scores and corresponding empirical *p*-values. The genes are ordered by absolute observed test scores. 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) [7] with additional confidence lines and genes marked as candidates, see Figure 2.1.

```
> expval$result[1:7, 1:5]

                observed   expected candidate       pvalue      qvalue
M84526_at       3.990578  1.1088376         1 1.402721e-07 0.0006197128
M27891_at       3.669657  0.9690305         1 2.805443e-07 0.0006197128
M89957_at      -3.153319 -1.1027192         1 4.208164e-07 0.0006197128
X82240_rna1_at -3.111376 -0.9680745         1 5.610885e-07 0.0006197128
U89922_s_at    -2.954233 -0.8948820         1 7.013606e-07 0.0006197128
M19507_at       2.925666  0.8960149         1 8.416328e-07 0.0006197128
M11722_at      -2.689999 -0.8422853         1 9.819049e-07 0.0006197128
```
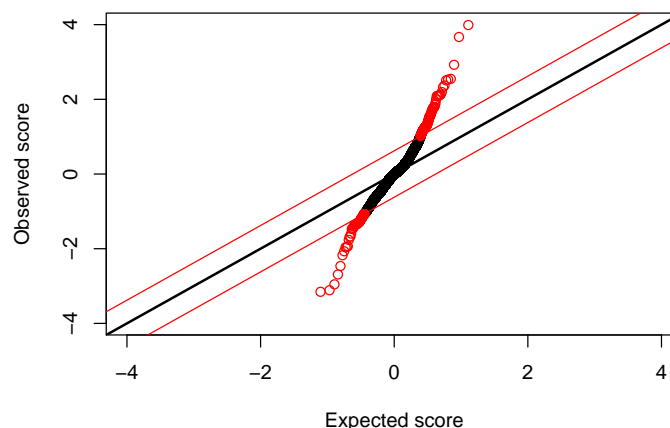
Figure 2.1: Expected versus observed test scores. Deviation from the diagonal line gives evidence for differential expression. The red lines mark the 95% confidence interval on the absolute difference between oberved and expected scores. The plotting call is `plot(expval,which="scores",grayscale=F,legend=F)`.

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) [6]. 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] 0.6197128
```

Column `result$index` contains the original gene ordering of the input object. With these numbers, resorting of the `result` table is possible without knowing the original order of the row names.
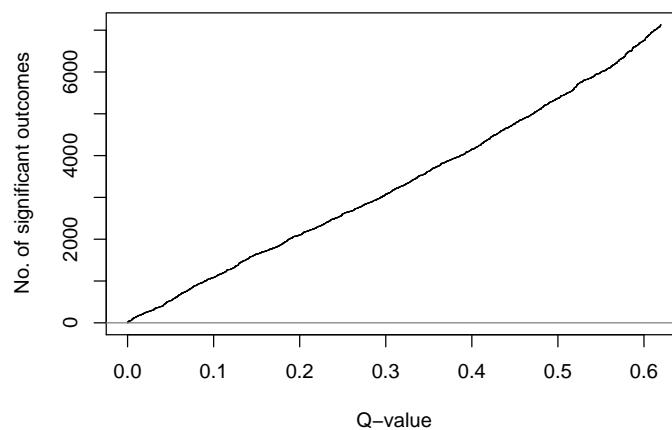
Figure 2.2: Stairplot of q-values against the resulting size of the list of significant genes. A list containing all genes with $q \leq q_0$ has an estimated global false discovery rate of q_0. The plotting call is `plot(expval,which="qvalues")`.

2.2   `twilight.pval`: Testing correlation

```
twilight.pval(xin, yin, method="fc", B=1000, yperm=NULL,
quant.ci=0.95, verbose=TRUE)
```

From version 1.1.0 on, function `twilight.pval` offers the computation of correlation scores instead of effect size scores. Now, vector `yin` can be any clinical parameter consisting of numerical values and having length equal to the number of samples. With `method="pearson"`, Pearson's coefficient of correlation to `yin` is computed for every gene in `xin`. With `method="spearman"`, `yin` and the rows of `xin` are converted into ranks and Spearman's rank correlation is computed.

Note that most input arguments of `twilight.pval` will be ignored. Only `B` takes effect and causes the computation of p-values based on `B` random permutations of `yin`. A matrix of user-specified permutations can be passed on using argument `yperm`. Here, each row has to contain a permutation of `yin`. Note that the values in `yperm` have to be changed to ranks beforehand if Spearman correlation is to be computed. Please note that the first row of `yperm` must be the input vector `yin` (probably changed into ranks). Otherwise, the p-value calculation will be incorrect.

All successive analyses like expected scores, p- and q-values are kept as before. As an illustration, we search for genes with high correlation to the highest scoring gene found in the effect size test. Figure 2.3 displays the resulting scores.

```
> gene <- exprs(golubNorm)[pval$result$index[1], ]
> corr <- twilight.pval(golubNorm, gene, method = "spearman",
+       quant.ci = 0.99, B = 100)

Compute vector of observed statistics.
Compute expected scores and p-values. This will take approx. 7 seconds.
Compute q-values.
Compute values for confidence lines.

> corr

 Twilight object with
     7129 transcripts
     observed and expected test statistics
     p- and q-values

 Estimated percentage of non-induced genes:
       pi0
0.6378229

 Function call:
 Test: spearman. Number of permutations: 100.
```
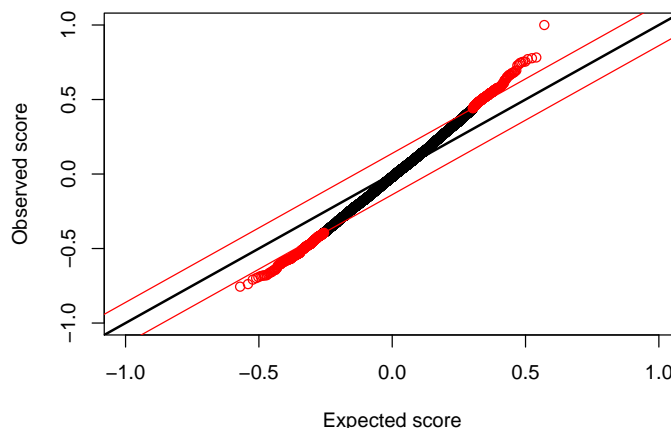
Figure 2.3: Expected versus observed Spearman correlation scores. Deviation from the diagonal line gives evidence for significant correlation. The red lines mark the 99% confidence interval on the absolute difference between oberved and expected scores. The plotting call is `plot(corr,which="scores",grayscale=F,legend=F)`.

Note that the overall percentage of non-induced genes π_0 is now interpreted as the overall percentage of genes not correlated to the clinical parameter under the null hypothesis.

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

```
                   observed    expected candidate       pvalue      qvalue
M27891_at         1.0000000   0.5703928         1 1.402721e-06 0.006378229
J03801_f_at       0.7822519   0.5403129         1 2.805443e-06 0.006378229
D88422_at         0.7772185   0.5239238         1 4.208164e-06 0.006378229
M19045_f_at       0.7713098   0.5083620         1 5.610885e-06 0.006378229
Z15115_at        -0.7557720  -0.5704060         1 7.013606e-06 0.006378229
M83667_rna1_s_at  0.7544589   0.4990852         1 8.416328e-06 0.006378229
M63138_at         0.7533647   0.4908699         1 9.819049e-06 0.006378229
X64072_s_at       0.7485502   0.4845475         1 1.122177e-05 0.006378229
M33195_at         0.7457052   0.4796520         1 1.262449e-05 0.006378229
U22376_cds2_s_at -0.7387023  -0.5409804         1 1.402721e-05 0.006378229
```

2.3 `twilight.filtering`: Filtering permutations

```
twilight.pval(..., filtering = FALSE)

twilight.filtering(xin, yin, method = "fc", paired = FALSE, s0 = 0,
verbose = TRUE, num.perm = 1000, num.take = 50)
```

From version 1.2.0 on, we included an permutation filtering algorithm. In a permutation based test approach, each permutation of the given class labels is thought to reflect the complete null model. However, in applications to real biological data, we often observe that certain permutations produce score distributions that still have larger margins than expected. Therefore, we treat each permutation as the original labeling, transform the permutation scores to pooled p-values and test the resulting distribution for uniformity. In an iterative search, we filter for a set of permutations whose p-value distributions fit well to a uniform distribution.

The filtering is added as an optional argument in function `twilight.pval`. Although large parts of the algorithm are written in C, the filtering is still time-consuming. Therefore, the default within `twilight.pval` is set to `FALSE`. If `filtering=TRUE`, the filtering is called internally with all the test parameters as given by the user. The only exception is the balancing parameter: The filtering is done on unbalanced permutations however `balance` is specified. Balancing is just a simpler way to select a set of permutations that are not too close to the given labeling. However, this will not remove other sources of deviation from a complete null distribution as the filtering does.

Calling the filtering within `twilight.pval` is very convenient. If one wants to further examine the filtered permutations, function `twilight.filtering` can be called directly. Most input arguments equal those of function `twilight.pval`, see Sections 2.1 and 2.2 for details on the different methods and formats. Only the fugde factor `s0` differs. It takes effect only if `method="z"` and is computed as the median pooled standard deviation if `s0=0`.

The two input arguments `num.perm` and `num.take` are important. The first one is the number of wanted permutations. Within `twilight.pval`, it is set to `B`. The argument `num.take` specifies the number of valid permutations that are kept in each step of the iteration. Within each step, this number increases by `num.take`. Hence, `num.take` might be chosen such that `num.take` is a divisor of `num.perm`. Within `twilight.pval`, `num.take` is set to the minimum of 50 and `num.perm`/20.

The output of function `twilight.filtering` is a list with two elements. The first element, `yperm`, is a matrix with the filtered permutations of `yin` in rows. The number of rows is approximately `num.perm`. The permutations are checked for uniqueness. If the number of possible unique permutations is less than `num.perm`, the algorithm stops earlier. In this case, the result is likely to be just the set of all possible permutations and it is not sure whether all of these really produce uniform p-value distributions. Here, it is advisable to lower `num.perm`.

The format of the output object `yperm` complies with the needed format of the input argument `yperm` in function `twilight.pval` where two-condition labels are binarized or

numerical values are changed to ranks if `method="spearman"`. Please note that the first row of `yperm` always contains the original labeling `yin` to be consistent with the other permutation functions described in Section 2.5.

The second output element is the numerical value `test`. For each filtered permutation, the Hamming distance to the original labeling `yin` is computed. The resulting distribution of Hamming distances is tested for randomness using a χ^2-test against the expected frequencies of Hamming distances resulting from all possible permutations of `yin`. The resulting χ^2 p-value is stored in `test`. A significant p-value gives evidence that the filtering did *not* leave a random subset of possible permutations. However, an effective run of `twilight.filtering` results in a *non*-significant χ^2 p-value `test`. Within `twilight.pval`, the result of the χ^2 test is reported in the output element `call` if `filtering=TRUE`.

As an illustration, we proceed with a quick example of permutation filtering. We perform the filtering on log ratio scores and only filter for 50 permutations in steps of 10.

```
> a <- twilight.filtering(golubNorm, id, method = "fc", num.perm = 50,
+     num.take = 10)

Filtering: Wait for 5 to 15 dots .....done

> dim(a$yperm)

[1] 50 38

> a$test

[1] 0.1875492
```

The filtering leads to a random subset of possible permutations. Next, we check whether one of these permutations really produces a uniform p-value distribution. As the first row of `yperm` has to contain the labeling for which the p-values will be computed, we have to remove the current first row which is the original labeling `yin`. Thus, we compute p-values for the first "random" permutation. The resulting histogram is shown in Figure 2.4.

```
> a$yperm <- a$yperm[-1, ]
> b <- twilight.pval(golubNorm, a$yperm[1, ], method = "fc",
+     yperm = a$yperm)

Compute vector of observed statistics.
Compute expected scores and p-values. This will take approx. 11 seconds.
Compute q-values.
Compute values for confidence lines.

> hist(b$result$pvalue, col = "gray", br = 20)
```
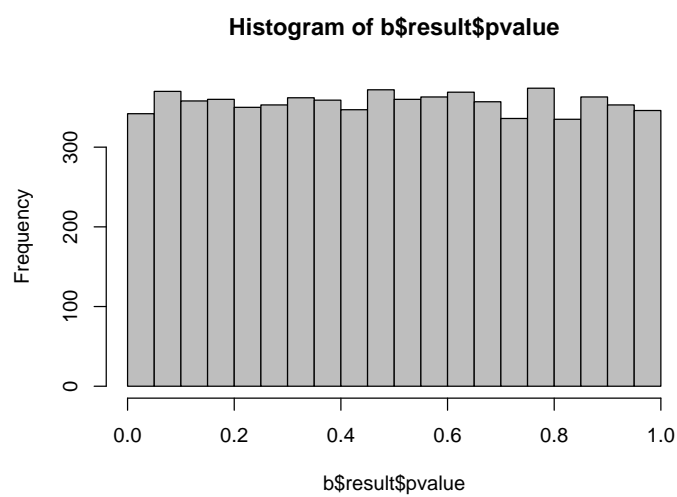
Figure 2.4: Histogram of p-values of one filtered permutation. The p-values are computed from pooling the scores of the set of filtered permutations. The resulting distribution appears to be uniform.

2.4 `twilight`: Estimating the local FDR

```
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.

Mapping scores to p-values allows to assume $f_0(p)$ 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 p-values into a uniformly distributed null part and an alternative part. The uniform part represents genes that are not differentially expressed. The proportion of the uniform part to the total number of genes in the experiment is a natural estimator for percentage π_0. We apply a smoothed density estimate based on the histogram counts of the observed mixture to estimate $f(p)$. Assuming uniformity leads to $f_0(p) = 1$ for all $p \in [0, 1]$. Hence, the ratio of the estimates $\widehat{\pi_0}$ and $\hat{f}(p)$ estimates the local false discovery rate for a certain p-value level.

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 p-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 the regularization parameter $\lambda \geq 0$. The regularization ensures that we find a separation such that the uniform part contains as many p-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 the 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 the 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 p-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


 Twilight object with
     7129 transcripts
     observed and expected test statistics
     p- and q-values
     local FDR
     bootstrap estimates of local FDR


 Bootstrap estimate of percentage of non-induced
 genes with lower and upper 95% CI:
       pi0 lower.pi0 upper.pi0
 0.6275649 0.5953114  0.656193


 Function call:
 Test: fc. Paired: FALSE. Number of permutations: 1000. Balanced: FALSE.
 Function twilight used lambda = 0.035


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


                     fdr    mean.fdr   lower.fdr   upper.fdr
M84526_at        0.01016839 0.01023125 0.006804447 0.01359302
M27891_at        0.01016949 0.01023236 0.006805623 0.01359414
M89957_at        0.01017059 0.01023347 0.006806799 0.01359526
X82240_rna1_at 0.01017169 0.01023457 0.006807976 0.01359638
U89922_s_at      0.01017279 0.01023568 0.006809152 0.01359750
```

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 p-values against the corresponding conditional probabilities of being induced given
the p-value level, that is $1 - \text{fdr}$, see Figure 2.5. Going back to observed scores, we produce
a *volcano plot*, that is observed scores versus local false discovery rate, see Figure 2.6.

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 p-value vector into a null and an alternative part. We estimate the
effect size distribution from the distribution of log ratio scores corresponding to p-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
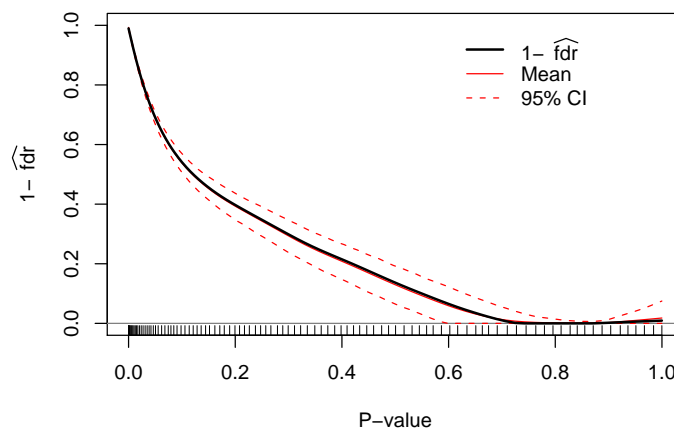
Figure 2.5: Curve of estimated local false discovery over p-values. The red lines denote the bootstrap mean (solid line) and the 95% bootstrap confidence interval on the local false discovery rate (dashed lines). The bottom ticks are 1% quantiles of p-values. The plotting call is `plot(exfdr,which="fdr",grayscale=F,legend=T)`.
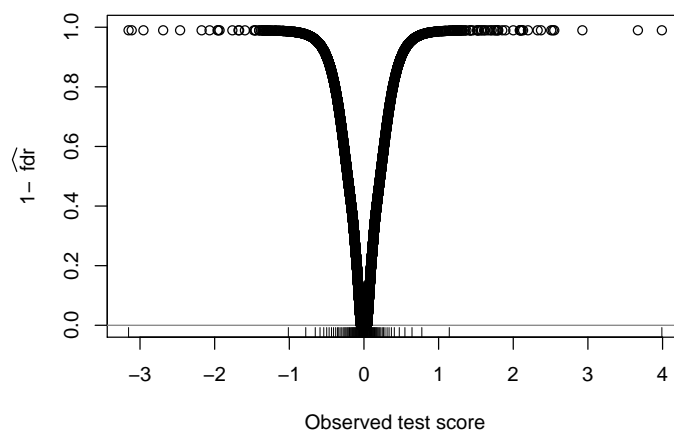
Figure 2.6: Volcano plot of observed test scores versus local false discovery rate. The bottom ticks are 1% quantiles of observed scores. The plotting call is `plot(exfdr,which="volcano")`.
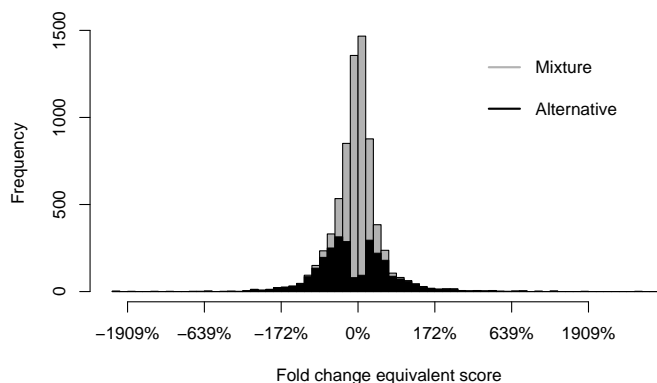
Figure 2.7: Observed effect size distribution (gray histogram) overlaid with the estimated effect size distribution under the null hypothesis (black histogram). The plotting call is `plot(exfdr,which="effectsize",legend=T)`.

with the averaged histogram of log ratios in the alternative, see Figure 2.7. 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 $(\exp(|F|)-1)\cdot\text{sign}(F)\cdot100\%$. 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, ]


      LogRatio Mixture Alternative
-2234%   -3.15       2         2.0
-2012%   -3.05       0         0.0
-1811%   -2.95       1         0.8
-1629%   -2.85       0         0.0
-1464%   -2.75       0         0.0
-1315%   -2.65       1         1.0
-1181%   -2.55       0         0.0
-1059%   -2.45       1         0.8
```

The input argument `clus` of function `twilight` is used to perform parallel computation within `twilight`. Parallelization saves computation time which is especially useful if the number of bootstrap samples B is large. With default `clus=NULL`, no parallelization 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.

2.5 `twilight.combi`: Enumerating permutations of binary vectors

```
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 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    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
```

Note that the function returns six balanced rows *and* the original input vector although it is not balanced.

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

Again, the input vector is part of the output.

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`. Similar to `twilight.combi`, these two functions return the input vector in the first row of their output matrices.

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

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

Chapter 3

Differences to earlier versions

3.1 Changes in version 1.6.1

It is now possible to directly compute observed test statistics via function `twilight.teststat`. Additional minor cosmetic changes in the plot function.

Within `twilight.pval`, the complete enumeration depends now on the value of argument B. If complete enumeration would lead to a larger number of permutations than B, it is not done but B random permutations are taken instead.

3.2 Changes in version 1.5.1 and 1.5.2

Minor cosmetic changes. The jump in version numbers is due to Bioconductor's version bumping regime for packages in the release and in the developmental repository.

3.3 Changes in version 1.2.3

We updated the bootstrapping procedure in `twilight.getlambda` to get a reliable value for the regularization parameter when π_0 is small and many genes are truly differentially expressed.

3.4 Changes in version 1.2.2

Changes in the C code which do not effect the results.

3.5 Changes in version 1.2.1

Bug fixed on the calculation of Hamming distances.

3.6   Changes in version 1.2.0

We added the argument `filtering = FALSE` to function `twilight.pval` which (if set to `TRUE`) invokes the filtering for class label permutations that produce uniform p-value distributions. The set of admissible permutations is found using function `twilight.filtering` which is called internally in function `twilight.pval`. However, it can also be used directly.

We changed the local FDR estimation in function `twilight` slightly. Instead of estimating both densities f_0 and f from the output of SEP, we rely on the uniform assumption such that $f_0(p) = 1$ for all $p \in [0, 1]$. Hence, the 10 runs of SEP lead to 10 estimates $\widehat{\pi_0}$. The average of these is taken as the final value which is mutliplied with the density estimate of the mixture density f.

The mixture density estimation of f also changed slightly. Still, the estimates are based on smoothed histogram counts. To improve the estimation for very small p-values, the histogram bins were changed from equidistant to quantile bins.

3.7   Changes in version 1.1.0

The computation of p-values in `twilight.pval` changed from gene-wise to pooled p-values. For the computation of a gene-wise p-value for gene i, only the permutation scores of gene i are taken into account. For pooled p-values, all permutation scores of all genes are taken as null distribution. This change has several advantages: First, gene-wise p-values were not monotonically increasing with scores because each gene had its own null distribution. Thus, two genes with almost equal scores might get quite different p-values. Now, the null distribution is the same for all genes, that is the union set of all permutation scores. Second, pooled p-values are less granular than gene-wise p-values. Gene-wise p-values are computed from `B` permutation scores whereas pooled p-values are computed from `B` · (number of genes) scores.

These two important features gave rise to further changes: The ordering of the `result` table is now more intuitive because the most significant genes on top have the highest scores, the lowest p- and q-values and are candidates (if there are any). In addition, the default value of the number of permutations `B` is lowered to 1000 permutations. Computation of pooled p-values is slower than for gene-wise p-values. On the other hand, changing to pooled p-values increases the number of values in the null distribution by the factor of number of genes. Hence, even with less permutations, the number of null values is larger than before.

We integrated Pearson and Spearman correlation coefficients into `twilight.pval`. Each gene is correlated to an numerical input vector. Expected scores are computed from random permutations of the input vector.

The `result` table contains an additional `index` column with genes indices which comes in handy for sorting back to original ordering.

All output matrices of the permutation functions `twilight.combi`, `twilight.permute.pair` and `twilight.permute.unpair` have the original labeling vector as first row. This is also

the case if balanced permutations are wanted, although the input vector is not balanced. Hence, the permutation matrix within `twilight.pval` now includes the original labeling even for balanced permutations implying that the smallest *possible p*-value is 1/(number of permutations).

3.8 Changes in version 1.0.3

A `print.twilight` function was added which produces a short information about the contents stored in the `twilight` object.

3.9 Changes in version 1.0.2

The `which` argument of the plot command changed from `plot1` style to more intuitive labels like `scores` or `fdr`.

Chapter 4

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] S. Scheid and R. Spang, "twilight; a Bioconductor package for estimating the local false discovery rate", *Bioinformatics*, vol. 21, no. 12, pp. 2921-2922, 2005.

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

[7] 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.