Random Rotation Package Introduction

Peter Hettegger

2023-10-24

Contents

1 Introduction . . . 2
2 Installation . . . 2
3 Sample dataset . . . 2
4 Quick start - linear models with batch effect correction . . . 3
5 Basic principle of random rotation methods . . . 7
6 Batch effect correction with subsequent linear model analysis . . . 9
6.1 Skewed null distribution of p values . . . 9
6.2 Unskewed p-values by random rotation . . . 13
6.3 Resampling based FDR . . . 16
6.4 Contrasts . . . 17
7 How many rotations ? . . . 20
8 Correlation matrices with non-block design . . . 22
9 Batch effect correction with linear mixed models . . . 22
9.1 Sample dataset . . . 22
9.2 Estimation of `cormat` . . . 23
9.3 Random rotation . . . 24
10 Session info . . . 26
References . . . 27

1 Introduction

`randRotation` is an R package intended for generation of randomly rotated data to resample null distributions of linear model based dependent test statistics. See also (Yekutieli and Benjamini 1999) for resampling dependent test statistics. The main application is to resample test statistics on linear model coefficients following arbitrary batch effect correction methods, see also section Quick start. The random rotation methodology is thereby applicable for linear models in combination with normally distributed data. Note that the resampling procedure is actually based on random orthogonal matrices, which is a broader class than random rotation matrices. Nevertheless, we adhere to the naming convention of (Langsrud 2005) designating this approach as random rotation methodology. The methodology used in this vignette is described in (Hettegger, Vierlinger, and Weinhaeusel 2021). Possible applications for resampling by rotation, that are outlined in this document, are: (i) linear models in combination with practically arbitrary (linear or non-linear) batch effect correction methods, section 6; (ii) generation of resampled datasets for evaluation of data analysis pipelines, section 6.2; (iii) calculation of resampling based test statistics for calculating resampling based p-values and false discovery rates (FDRs), sections 6.2 and 6.3; and (iv) batch effect correction with linear mixed models 9.

Generally, the rotation approach provides a methodology for generating resampled data in the context of linear models and thus potentially has further conceivable areas of applications in high-dimensional data analysis with dependent variables. Nevertheless, we focus this document on the outlined range of issues in order to provide an intuitive and problem-centered introduction.

2 Installation

Execute the following code to install package `randRotation`:

```
if(!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("randRotation")
```

3 Sample dataset

For subsequent analyses we create a hypothetical dataset with 3 batches, each containing 5 Control and 5 Cancer samples with 1000 features (genes). Note that the created dataset is pure noise and no artificial covariate effects are introduced. We thus expect uniformly distributed p-values for linear model coefficients.

```
library(randRotation)
set.seed(0)
# Dataframe of phenotype data (sample information)
pdata <- data.frame(batch = as.factor(rep(1:3, c(10,10,10))),
                    phenotype = rep(c("Control", "Cancer"), c(5,5)))
features <- 1000

# Matrix with random gene expression data
edata <- matrix(rnorm(features * nrow(pdata)), features)
rownames(edata) <- paste("feature", 1:nrow(edata))
```

```
xtabs(data = pdata)
#>      phenotype
#> batch Cancer Control
#>     1      5       5
#>     2      5       5
#>     3      5       5
```

4 Quick start - linear models with batch effect correction

A main application of the package is to resample null distributions of parameter estimates for linear models following batch effect correction. We first create our model matrix:

```
mod1 <- model.matrix(~1+phenotype, pdata)
head(mod1)
#>   (Intercept) phenotypeControl
#> 1           1                1
#> 2           1                1
#> 3           1                1
#> 4           1                1
#> 5           1                1
#> 6           1                0
```

We then initialise the random rotation object with `initBatchRandrot` and select the `phenotype` coefficient as the null hypothesis coefficient:

```
rr <- initBatchRandrot(Y = edata, X = mod1, coef.h = 2, batch = pdata$batch)
#> Initialising batch "1"
#> Initialising batch "2"
#> Initialising batch "3"
```

Now we define the data analysis pipeline that should be run on the original dataset and on the rotated dataset. Here we include as first step (I) our batch effect correction routine `ComBat` (*sva* package) and as second step (II) we obtain the t-values for covariate `phenotype` from the linear model fit.

```
statistic <- function(Y, batch, mod){
  # (I) Batch effect correction with "Combat" from the "sva" package
  Y <- sva::ComBat(dat = Y, batch = batch, mod = mod)

  # (II) Linear model fit
  fit1 <- limma::lmFit(Y, design = mod)
  fit1 <- limma::eBayes(fit1)
  abs(fit1$t[,2])
}
```

Note that larger values of the `statistic` function are considered as more significant in the subsequently used `pFdr` function. We thus take the absolute values of the coefficients in order to calculate two-sided (two-tailed) p-values with `pFdr`. We emphasize that we highly

recommend using scale independent statistics (pivotal quantities) as e.g. t-values instead of parameter estimates (as with `coef`), see also `?randRotation::pFdr`. The explicit function calls like `sva::ComBat` are required if parallel computing is used, see `?randRotation::rotateStat`.

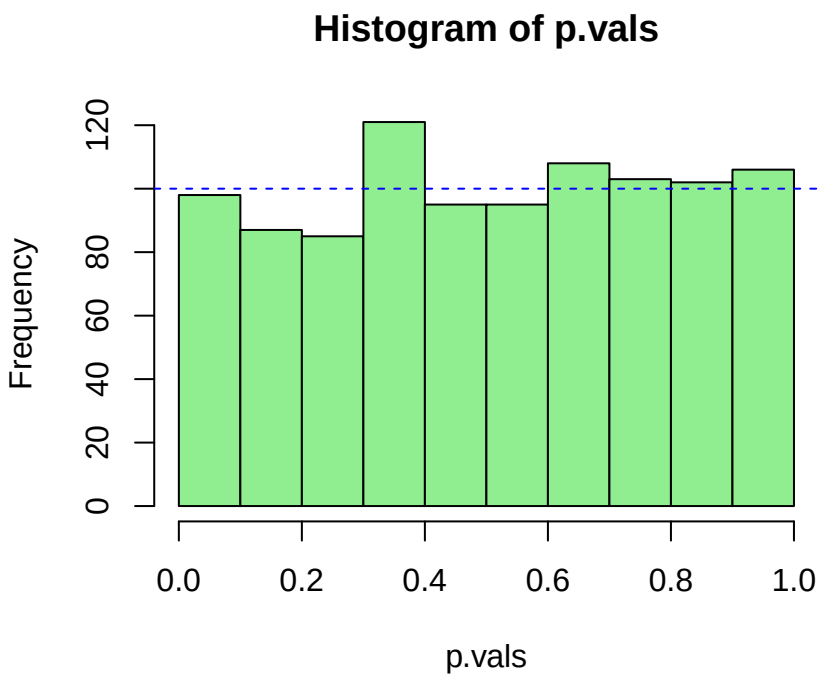
The `rotateStat` function calculates `statistic` on the original (non-rotated) dataset and on 10 random rotations. `batch` and `mod` are provided as additional parameters to `statistic`.

```
rs1 <- rotateStat(initialised.obj = rr, R = 10, statistic = statistic,
                  batch = pdata$batch, mod = mod1)
```

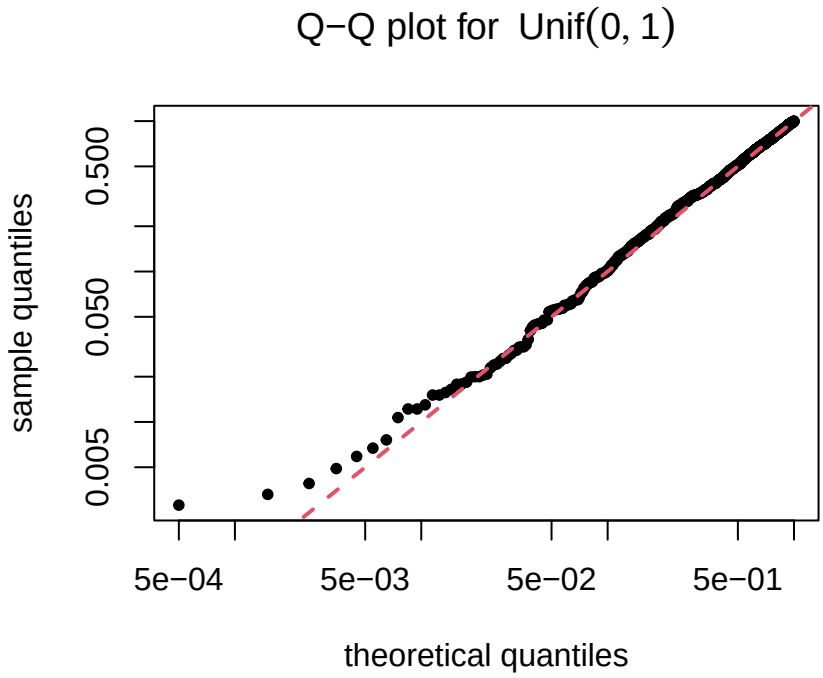
```
rs1
#> Rotate stat object
#>
#> R = 10
#>
#> dim(s0): 1000 1
#>
#> Statistic:
#> function(Y, batch, mod){
#>   # (I) Batch effect correction with "Combat" from the "sva" package
#>   Y <- sva::ComBat(dat = Y, batch = batch, mod = mod)
#>
#>   # (II) Linear model fit
#>   fit1 <- limma::lmFit(Y, design = mod)
#>   fit1 <- limma::eBayes(fit1)
#>   abs(fit1$t[,2])
#> }
#> <bytecode: 0x7f7858b8a7f8>
#>
#> Call:
#> rotateStat(initialised.obj = rr, R = 10, statistic = statistic,
#>     batch = pdata$batch, mod = mod1)
```

Resampling based p-values are obtained with `pFdr`. As we use "pooling" of the rotated statistics in `pFdr`, 10 random rotations are sufficient.

```
p.vals <- pFdr(rs1)
hist(p.vals, col = "lightgreen");abline(h = 100, col = "blue", lty = 2)
```




```
qqunif(p.vals)
```

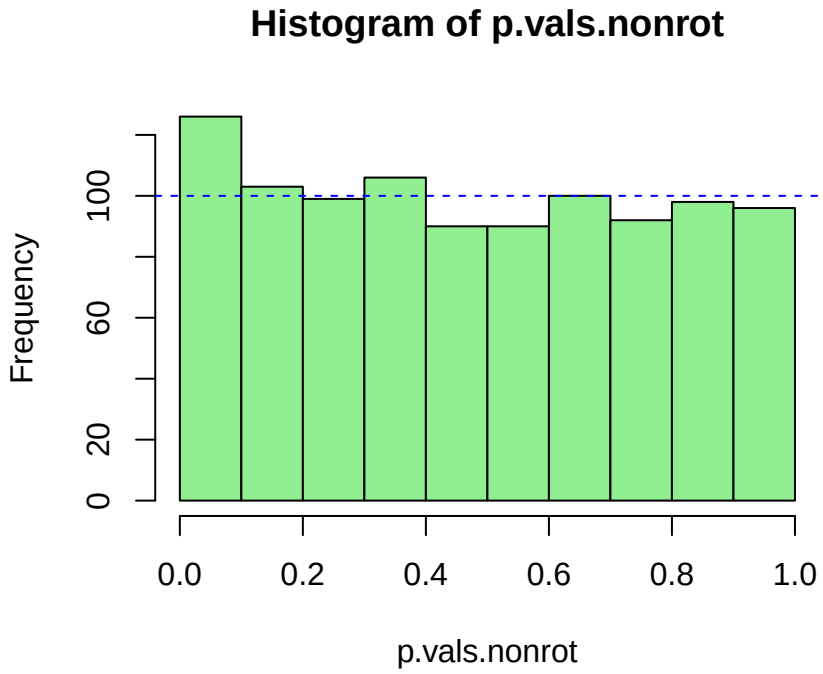


We see that, as expected, our p-values are approximately uniformly distributed.

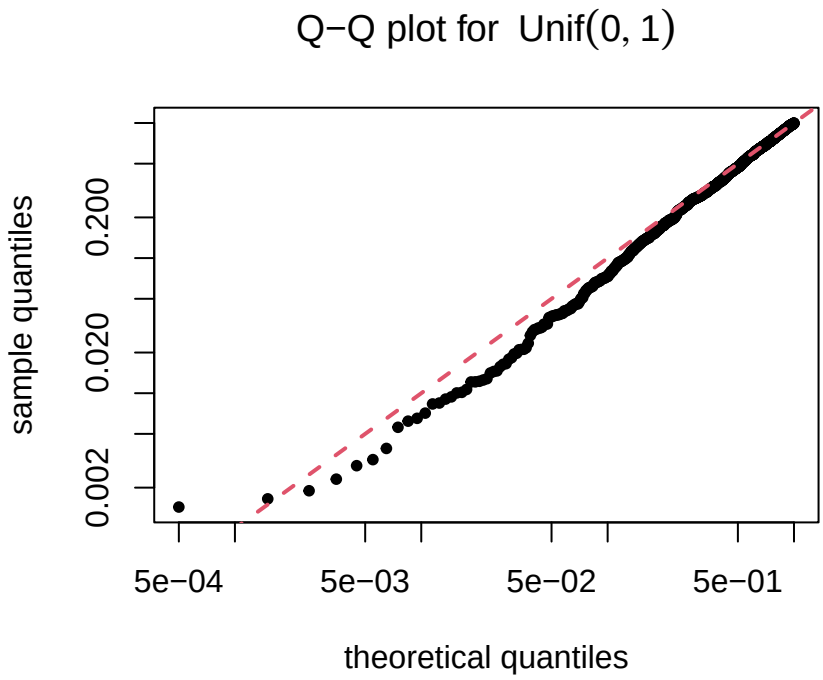
Hint: The outlined procedure also works with `statistic` functions which return multiple columns (`rotateStat` and `pFdr` handle functions returning multiple columns adequately). So one could e.g. perform multiple batch effect correction methods and calculate the statistics of interest for each correction method. By doing this, one could subsequently evaluate the influence of different batch effect correction methods on the statistic of interest.

Additional info: Below, the analysis pipeline is performed without rotation for comparison with the previous analyses. Following batch effect correction with `ComBat` (*sva* package), we obtain p-values from linear fit coefficients (using the *limma* package) as follows:

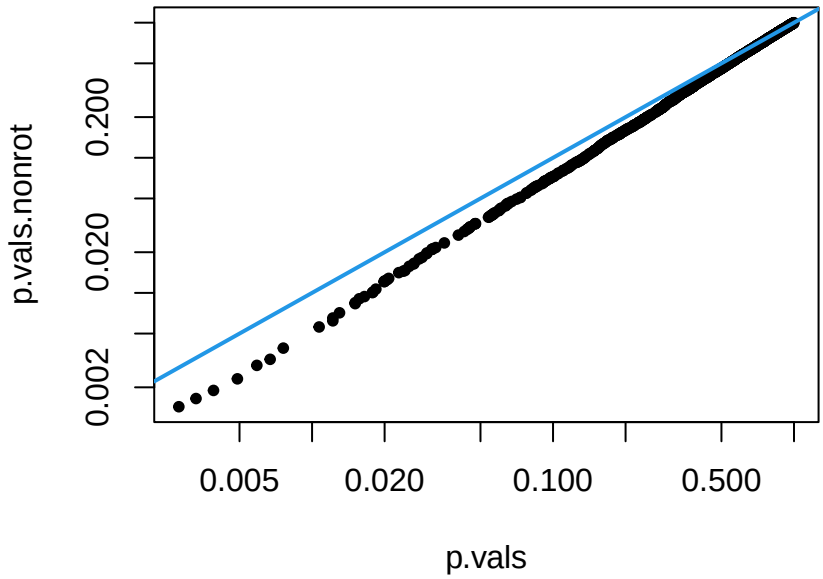
```
library(limma)
library(sva)
#> Loading required package: mgcv
#> Loading required package: nlme
#> This is mgcv 1.9-0. For overview type 'help("mgcv-package")'.
#> Loading required package: genefilter
#> Loading required package: BiocParallel

edata.combat <- ComBat(dat = edata, batch = pdata$batch, mod = mod1)
#> Found3batches
#> Adjusting for1covariate(s) or covariate level(s)
#> Standardizing Data across genes
#> Fitting L/S model and finding priors
#> Finding parametric adjustments
#> Adjusting the Data
fit1 <- lmFit(edata.combat, mod1)
fit1 <- eBayes(fit1)

# P-values from t-statistics
p.vals.nonrot <- topTable(fit1, coef = 2, number = Inf, sort.by="none")$P.Value

hist(p.vals.nonrot, col = "lightgreen");abline(h = 100, col = "blue", lty = 2)
```




```
qqunif(p.vals.nonrot)
```




```
plot(p.vals, p.vals.nonrot, log = "xy", pch = 20)
abline(0,1, col = 4, lwd = 2)
```



We see that the p-values are non-uniformly distributed. See also section 6.1.

5 Basic principle of random rotation methods

In the random rotation methodology, the observed data vectors (for each feature) are rotated in way that the *determined* coefficients ($\boldsymbol{B_D}$ in Langsrud (2005)) stay constant when resampling under the null hypothesis $H_0 : \boldsymbol{B_H} = \boldsymbol{0}$, see (Langsrud 2005).

The following example shows that the intercept coefficient of the *null model* does not change when rotation is performed under the null hypothesis:

```
# Specification of the full model
mod1 <- model.matrix(~1+phenotype, pdata)

# We select "phenotype" as the coefficient associated with H0
# All other coefficients are considered as "determined" coefficients
rr <- initRandrot(Y = edata, X = mod1, coef.h = 2)

coefs <- function(Y, mod){
  t(coef(lm.fit(x = mod, y = t(Y))))
}

# Specification of the H0 model
mod0 <- model.matrix(~1, pdata)

coef01 <- coefs(edata, mod0)
coef02 <- coefs(randrot(rr), mod0)

head(cbind(coef01, coef02))
#>           (Intercept) (Intercept)
#> feature 1    0.040777    0.040777
#> feature 2   -0.001669   -0.001669
#> feature 3    0.036254    0.036254
#> feature 4   -0.272032   -0.272032
#> feature 5    0.105839    0.105839
#> feature 6   -0.012137   -0.012137

all.equal(coef01, coef02)
#> [1] TRUE
```

However, the coefficients of the *full model* do change (for this parametrisation) when rotation is performed under the null hypothesis:

```
coef11 <- coefs(edata, mod1)
coef12 <- coefs(randrot(rr), mod1)

head(cbind(coef11, coef12))
#>           (Intercept) phenotypeControl (Intercept) phenotypeControl
#> feature 1    0.236258         -0.39096    -0.30040          0.68235
#> feature 2    0.023970         -0.05128     0.15804         -0.31942
#> feature 3    0.180283         -0.28806    -0.05786          0.18823
#> feature 4   -0.007109         -0.52984    -0.25566         -0.03275
#> feature 5    0.452219         -0.69276     0.09968          0.01232
#> feature 6    0.031197         -0.08667    -0.12935          0.23442
```

This is in principle how resampling based tests are constructed. Note that the change in both coefficients is due to parametrisation of the model. Compare e.g. the following parametrisation, where the *determined* coefficient (Intercept) does not change:

```
mod2 <- mod1
mod2[,2] <- mod2[,2] - 0.5

coef11 <- coefs(edata, mod2)
coef12 <- coefs(randrot(rr), mod2)

head(cbind(coef11, coef12))
#>           (Intercept) phenotypeControl (Intercept) phenotypeControl
#> feature 1    0.040777         -0.39096    0.040777           0.2885
#> feature 2   -0.001669         -0.05128   -0.001669          -0.4767
#> feature 3    0.036254         -0.28806    0.036254          -0.5369
#> feature 4   -0.272032         -0.52984   -0.272032          -0.2712
#> feature 5    0.105839         -0.69276    0.105839          -0.2143
#> feature 6   -0.012137         -0.08667   -0.012137           0.2032
```

6 Batch effect correction with subsequent linear model analysis

In the following we outline the use of the `randRotation` package for linear model analysis following batch effect correction as a prototype application in current biomedical research. We highlight the problems faced when batch effect correction is separated from data analysis with linear models. Although data analysis procedures with combined batch effect correction and model inference should be preferred, the separation of batch effect correction from subsequent analysis is unavoidable for certain applications. In the following we use `ComBat` (*sva* package) as a model of a "black box" batch effect correction procedure. Subsequent linear model analysis is done with the *limma* package. We use `limma` and `ComBat` as model functions for demonstration, as these are frequently used in biomedical research. We want to emphasize that neither the described issues are specific to these functions, nor do we want to somehow defame these highly useful packages.

6.1 Skewed null distribution of p values

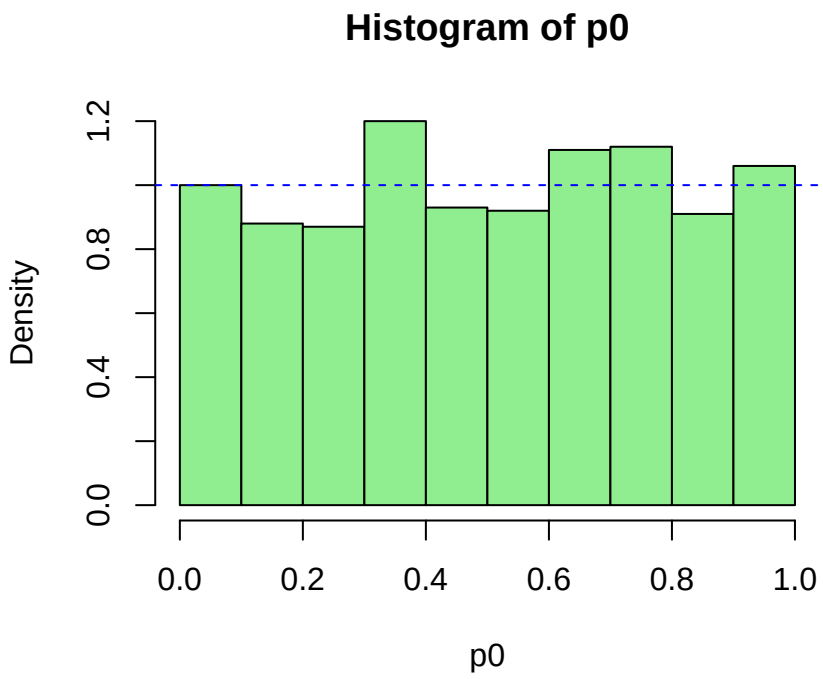
Separating a (possibly non-linear) batch effect correction method from linear model analysis could practically lead to non-uniform (skewed) null distributions of p-values for testing linear model coefficients. The intuitive reason for this skew is that the batch effect correction method combines information of all samples to remove the batch effects. After removing the batch effects, the samples are thus no longer independent. For further information please refer to section df estimation and to the references.

The following example demonstrates the influence of the batch effect correction on the distribution of p-values. We first load the *limma* package and create the model matrix with the intercept term and the phenotype term.

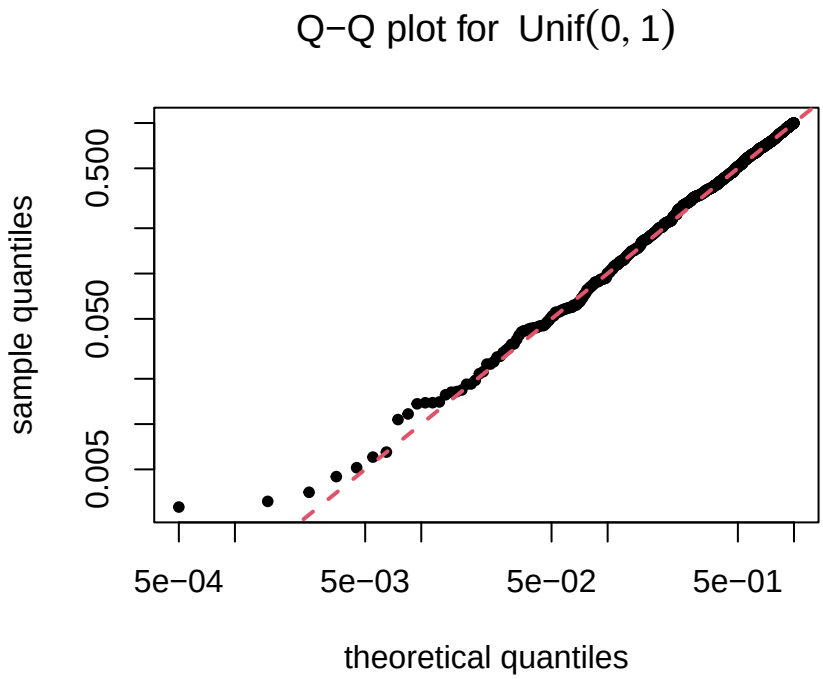
```
library(limma)

mod1 = model.matrix(~phenotype, pdata)
```

Remember that our sample dataset is pure noise. Thus, without batch effect correction, fitting a linear model with `limma` and testing the phenotype coefficient results in uniformly distributed p-values:

```
# Linear model fit
fit0 <- lmFit(edata, mod1)
fit0 <- eBayes(fit0)

# P values for phenotype coefficient
p0 <- topTable(fit0, coef = 2, number = Inf, adjust.method = "none",
               sort.by = "none")$P.Value
hist(p0, freq = FALSE, col = "lightgreen", breaks = seq(0,1,0.1))
abline(1,0, col = "blue", lty = 2)
```




```
qqunif(p0)
```

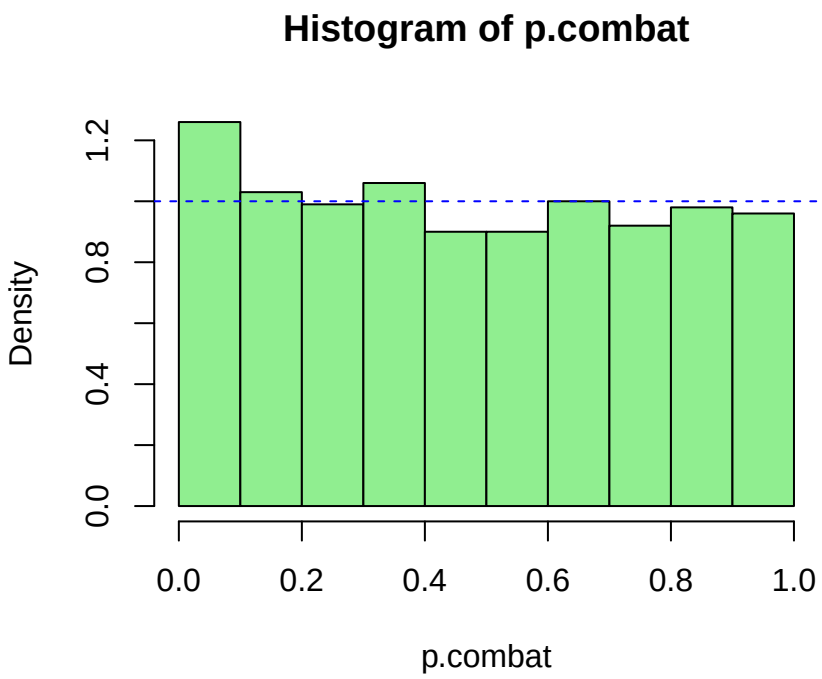


We now perform batch effect correction using `ComBat` (*sva* package):

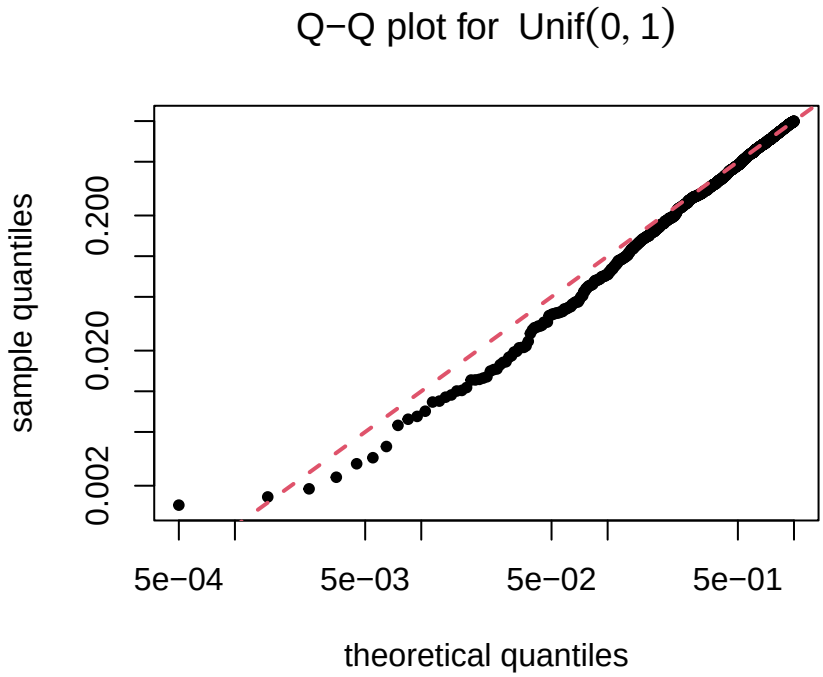
```
library(sva)
edata.combat = ComBat(edata, batch = pdata$batch, mod = mod1)
#> Found3batches
#> Adjusting for1covariate(s) or covariate level(s)
#> Standardizing Data across genes
#> Fitting L/S model and finding priors
#> Finding parametric adjustments
#> Adjusting the Data
```

Performing the model fit and testing the phenotype effect on this modified dataset results in a skewed p-value distribution:

```
# Linear model fit
fit1 <- lmFit(edata.combat, mod1)
fit1 <- eBayes(fit1)

# P value for phenotype coefficient
p.combat <- topTable(fit1, coef = 2, number = Inf, adjust.method = "none",
                     sort.by = "none")$P.Value
hist(p.combat, freq = FALSE, col = "lightgreen", breaks = seq(0,1,0.1))
abline(1,0, col = "blue", lty = 2)
```




```
qqunif(p.combat)
```



The histogram and Q-Q plot clearly show that the null-distribution of p-values is skewed when linear model analysis is performed following batch effect correction in a data analysis pipeline of this type. This problem is known and described e.g. in (Nygaard, Rødland, and Hovig 2015). Note that the null-distribution is skewed although the experimental design is balanced.

6.2 Unskewed p-values by random rotation

In the following, we take the data analysis pipeline of the previous section and incorporate it into the random rotation environment. The `initBatchRandrot` function initialises the random rotation object with the design matrix of the linear model. We thereby specify the coefficients associated with the null hypothesis H_0 (see also 5) with `coef.h`. Additionally, the batch covariate is provided.

Note that the implementation with `initBatchRandrot` in principle implicitly assumes a block design of the correlation matrix and restricted roation matrix, see also @ref{nonblock}.

```
init1 <- initBatchRandrot(edata, mod1, coef.h = 2, batch = pdata$batch)
#> Initialising batch "1"
#> Initialising batch "2"
#> Initialising batch "3"
```

We now pack the data analysis pipeline of above into our statistic function, which is called for the original (non-rotate) data and for all data rotations:

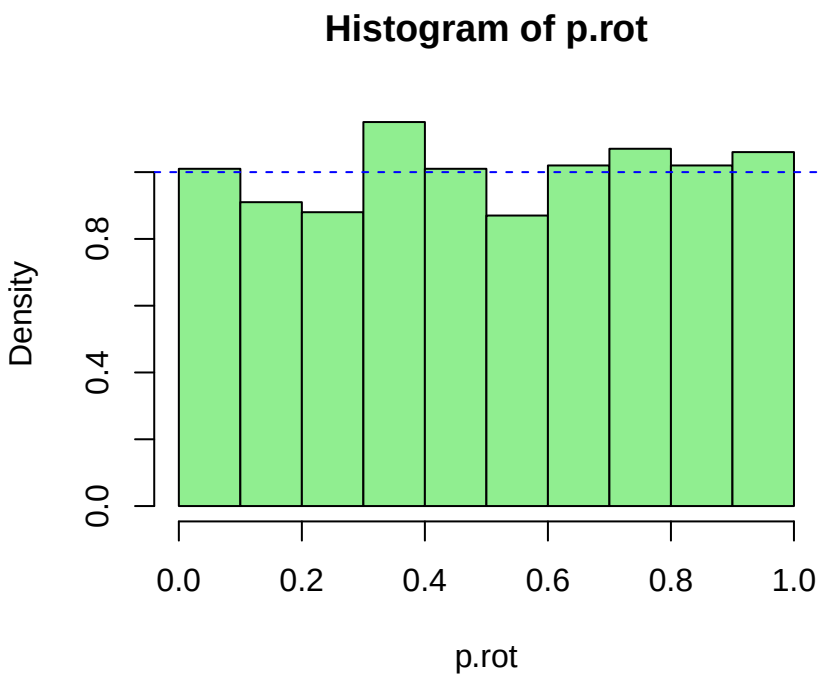
```
statistic <- function(Y, batch, mod, coef){
  Y.tmp <- sva::ComBat(dat = Y, batch = batch, mod = mod)

  fit1 <- limma::lmFit(Y.tmp, mod)
  fit1 <- limma::eBayes(fit1)
  # The "abs" is needed for "pFdr" to calculate 2-tailed statistics
  abs(fit1$t[,coef])
}
```

Data rotation and calling the `statistic` function is performed with `rotateStat`.

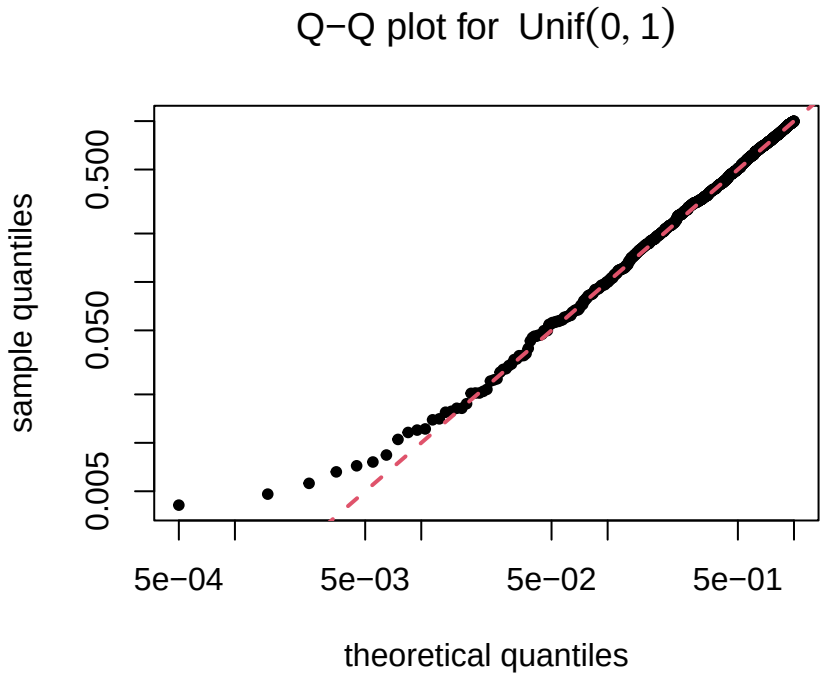
```
res1 <- rotateStat(initialised.obj = init1, R = 10, statistic = statistic,
                   batch = pdata$batch, mod = mod1, coef = 2)
```

As we use pooling of rotated statistics, `R = 10` resamples should be sufficient (see also 7). We now calculate rotation based p-values with `pFdr`:

```
p.rot <- pFdr(res1)
head(p.rot)
#>              [,1]
#> feature 1 0.30397
#> feature 2 0.91951
#> feature 3 0.44076
#> feature 4 0.13709
#> feature 5 0.05759
#> feature 6 0.83342

hist(p.rot, freq = FALSE, col = "lightgreen", breaks = seq(0,1,0.1))
abline(1,0, col = "blue", lty = 2)
```



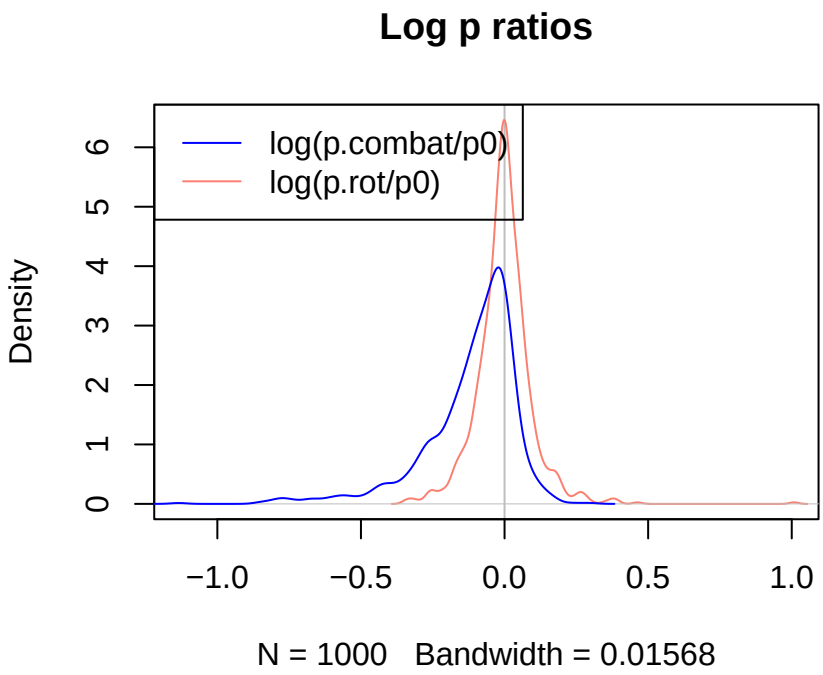

```
qqunif(p.rot)
```



We see that our rotated p-values are roughly uniformly distributed.

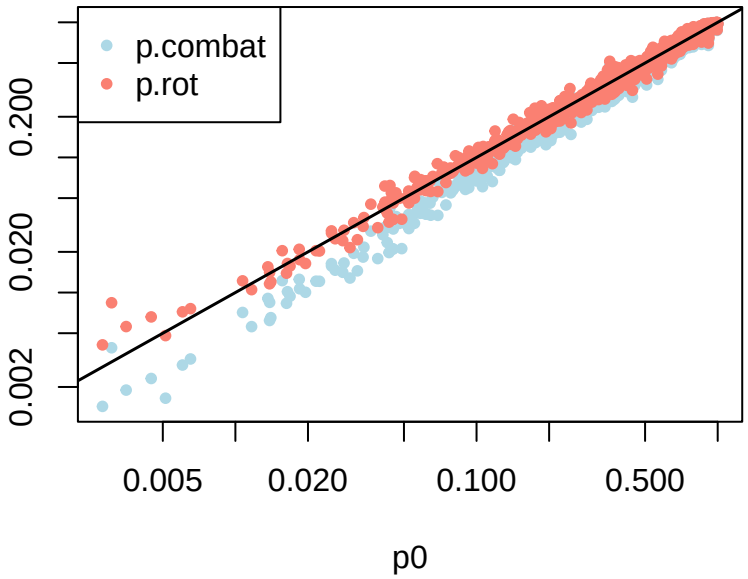
For illustration of the skewness of non-rotated p-values, we compare the non-rotated p-values `p.combat` (batch corrected), the rotated p-values `p.rot` (batch corrected) and the p-values from linear model analysis without batch correction `p0`.

```
plot(density(log(p.rot/p0)), col = "salmon", "Log p ratios",
     panel.first = abline(v=0, col = "grey"),
     xlim = range(log(c(p.rot/p0, p.combat/p0))))
lines(density(log(p.combat/p0)), col = "blue")
legend("topleft", legend = c("log(p.combat/p0)", "log(p.rot/p0)"),
       lty = 1, col = c("blue", "salmon"))
```



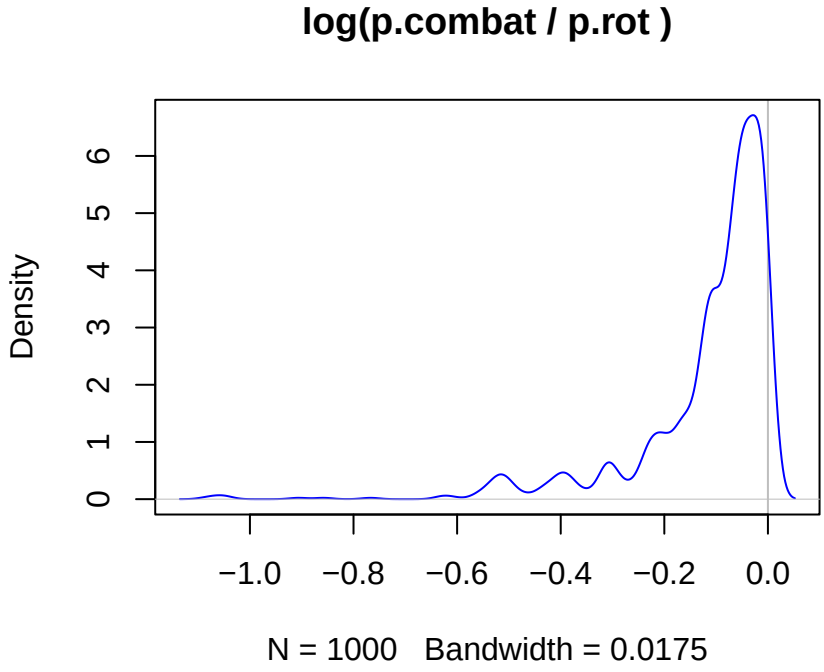
We see the skew of the non-rotated p-values towards lower values. This is also seen in another illustration below:

```
plot(p0, p.combat, log = "xy", pch = 20, col = "lightblue", ylab = "")
points(p0, p.rot, pch = 20, col = "salmon")
abline(0,1, lwd = 1.5, col = "black")
legend("topleft", legend = c("p.combat", "p.rot"), pch = 20,
       col = c("lightblue", "salmon"))
```



The non-rotated p-values are on average lower than the rotated p-values:

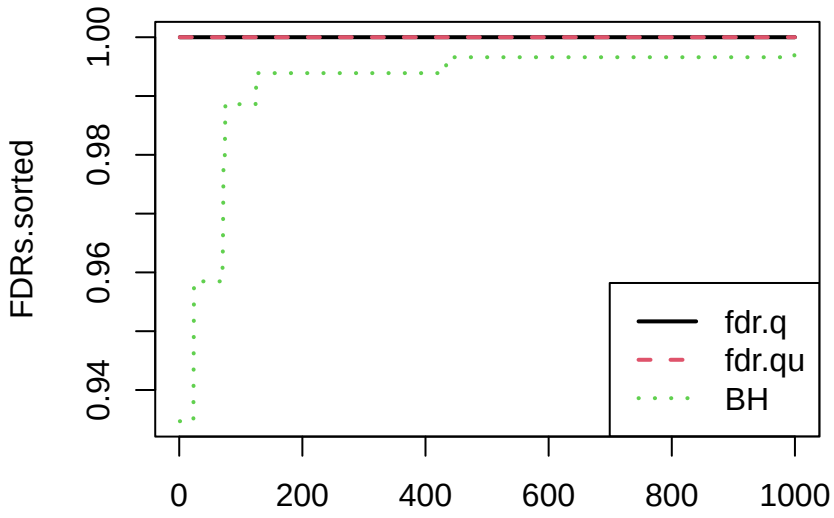
```
plot(density(log(p.combat/p.rot)), col = "blue",
     main = "log(p.combat / p.rot )", panel.first = abline(v=0, col = "grey"))
```



6.3 Resampling based FDR

Additionally to resampling based p-values, the method `pFdr` could also be used for estimating resampling based false discovery rates (Yekutieli and Benjamini 1999).

```
fdr.q  <- pFdr(res1, "fdr.q")
fdr.qu <- pFdr(res1, "fdr.qu")
fdr.BH <- pFdr(res1, "BH")

FDRs <- cbind(fdr.q, fdr.qu, fdr.BH)
ord1 <- order(res1$s0, decreasing = TRUE)

FDRs.sorted <- FDRs[ord1,]

matplot(FDRs.sorted, type = "l", lwd = 2)
legend("bottomright", legend = c("fdr.q", "fdr.qu", "BH"), lty = 1:5, lwd = 2,
       col = 1:6)
```




```
head(FDRs.sorted)
#>             [,1] [,2]   [,3]
#> feature 990    1    1 0.9347
#> feature 478    1    1 0.9347
#> feature 229    1    1 0.9347
#> feature 875    1    1 0.9347
#> feature 374    1    1 0.9347
#> feature 254    1    1 0.9347
```

6.4 Contrasts

The random rotation methodology can also be applied for contrasts. We introduce an artificial group effect between group 2 and group 3 for the first 100 features (we use that later in 7).

```
edata[,] <- rnorm(length(edata))
group <- as.factor(rep(1:3, 10))

# add group effect for the first 100 features
group.effect <- rep(c(0,0,1), 10)
edata[1:100,] <- t(t(edata[1:100,]) + group.effect)
```

```
mod.groups <- model.matrix(~ group)

contrasts1 <- limma::makeContrasts("2vs3" = group2 - group3,
                                   levels = mod.groups)
contrasts1
#>            Contrasts
#> Levels      2vs3
#>   Intercept    0
#>   group2       1
#>   group3      -1
```

Using `contrastModel` we transform our model matrix to a new model matrix (with same dimensions as `mod.groups`) which includes the contrast as last coefficient. Thereby, all contrasts are set as `coef.h` (in the attributes of `mod.cont`).

```
mod.cont <- contrastModel(X = mod.groups, C = contrasts1)
```

The random rotation object is automatically initalised with the contrasts set as `coef.h`:

```
init1 <- initBatchRandrot(edata, mod.cont, batch = pdata$batch)
#> Initialising batch "1"
#> Initialising batch "2"
#> Initialising batch "3"
```

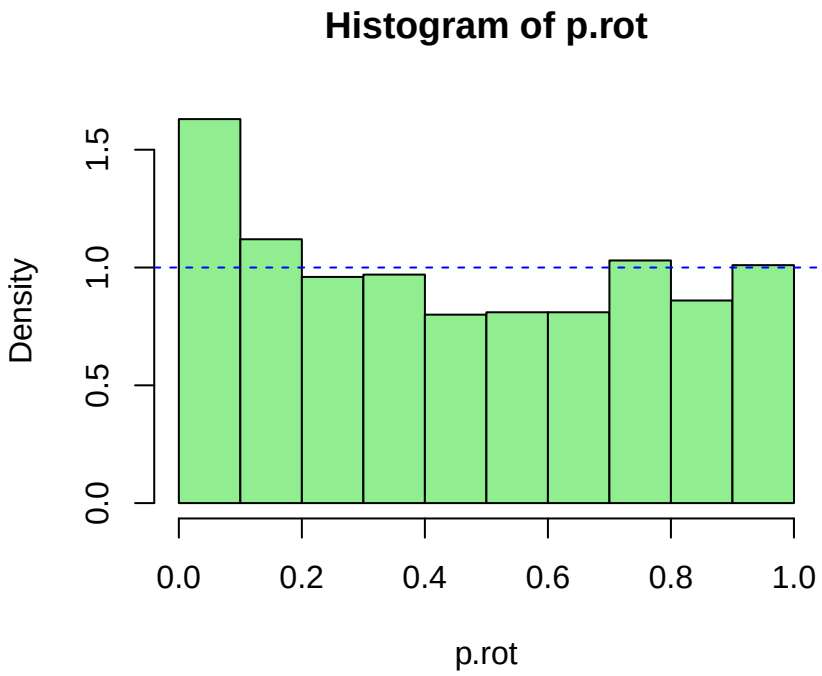
Similarly to above, we can now test our contrast in the batch effect adjusted data using random rotations:

```
statistic <- function(Y, batch, mod, cont){
  Y.tmp <- sva::ComBat(dat = Y, batch = batch, mod = mod)

  fit1 <- limma::lmFit(Y.tmp, mod)

  fit1 <- limma::contrasts.fit(fit1, cont)
  fit1 <- limma::eBayes(fit1)

  # The "abs" is needed for "pFdr" to calculate 2-tailed statistics
  abs(fit1$t[,1])
}

res1 <- rotateStat(initialised.obj = init1, R = 20, statistic = statistic,
                   batch = pdata$batch, mod = mod.groups, cont = contrasts1)
```

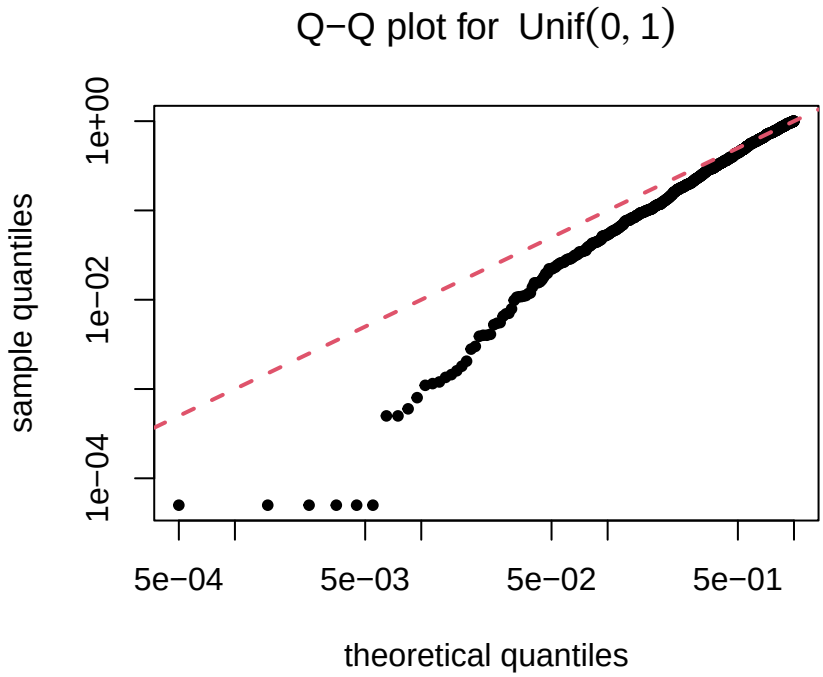
We calculate the rotation based p-values with `pFdr`:

```
p.rot <- pFdr(res1)
head(p.rot)
#>              [,1]
#> feature 1 0.01190
#> feature 2 0.17979
#> feature 3 0.00005
#> feature 4 0.01100
#> feature 5 0.05205
```

```
#> feature 6 0.04545

hist(p.rot, freq = FALSE, col = "lightgreen", breaks = seq(0,1,0.1))
abline(1,0, col = "blue", lty = 2)
```




```
qqunif(p.rot)
```



7 How many rotations ?

The sufficient number of rotations `R` for simulating the null-distribution of our statistic of interest depends on multiple factors and is different for each application. A possible guiding principle for finding a sufficient number of resamples could be the following.

Increase the number of resamples `R` until:

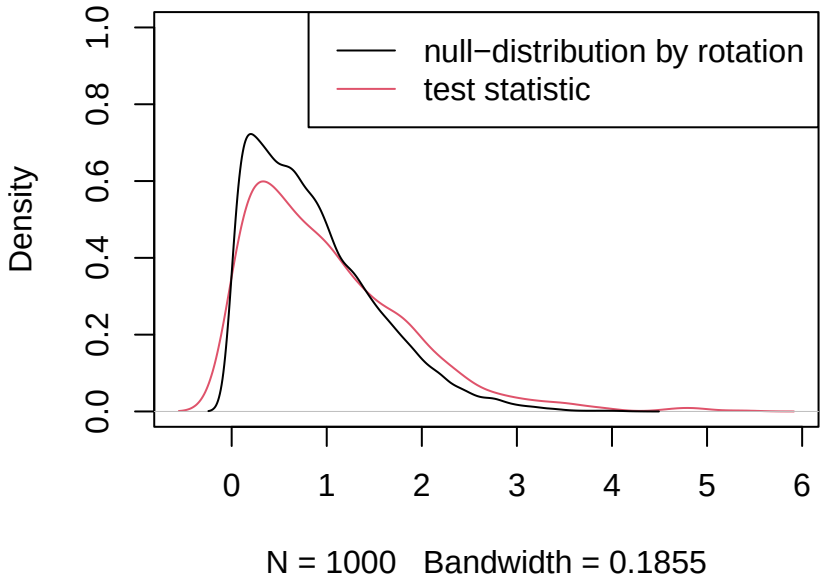
- the rotation procedure provides an adequately smooth null-distribution of the statistic of interest.
- the results (e.g. the number of features with `fdr < 0.05`) and/or null-distribution do not change substantially if the rotation procedure is repeated with the same `R`.

Consequently, `R` must be increased if one needs high precision in the tail regions of the null distribution (so e.g. if `fdr < 0.01` is used instead of `fdr < 0.05`). Nevertheless, note that the ordering of the features does not change if `R` is increased.

Large `R` might be required if e.g. features are highly dependent. In this case, for a single rotation, the resulting values of our statistic are highly similar and thus only *small intervals of the null-distribution* are simulated.

The following figure shows the null distribution (`R = 20`) and the test values of the example given in 6.4:

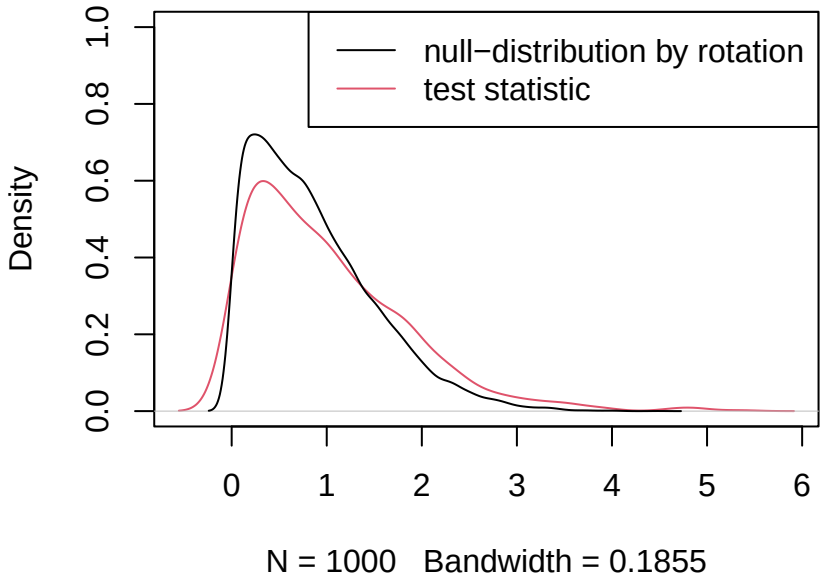
```
plot(density(res1$s0), main = "", ylim = c(0,1), col = 2)
lines(density(res1$stats[[1]]), col = 1)
legend("topright", col = 1:2, lty = 1,
       legend = c("null-distribution by rotation", "test statistic"))
```



We repeat the rotation procedure with `R = 20`:

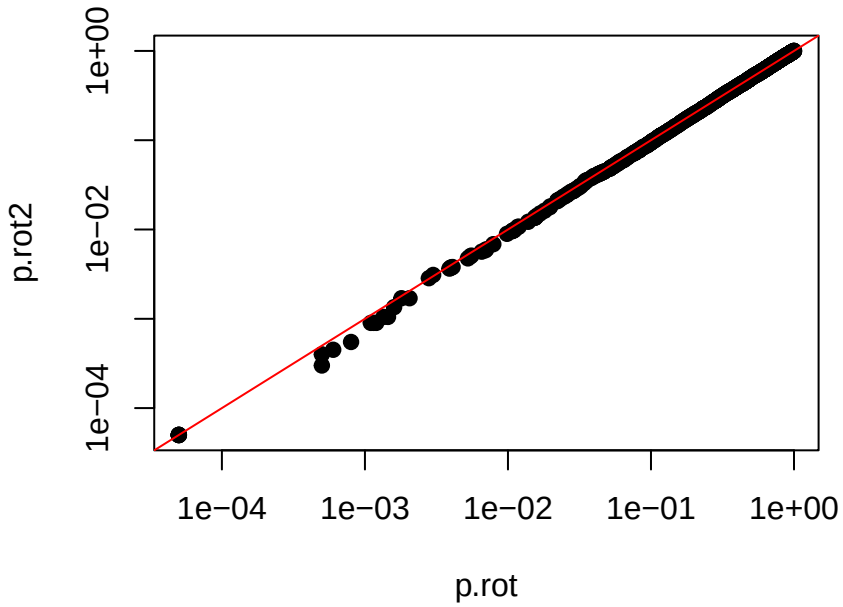
```
res2 <- rotateStat(initialised.obj = init1, R = 20, statistic = statistic,
                   batch = pdata$batch, mod = mod.groups, cont = contrasts1)

p.rot2 <- pFdr(res2)
```

```
plot(density(res2$s0), main = "", ylim = c(0,1), col = 2)
lines(density(res2$stats[[1]]), col = 1)
legend("topright", col = 1:2, lty = 1,
       legend = c("null-distribution by rotation", "test statistic"))
```



Comparing the p-values shows:

```
plot(p.rot, p.rot2, pch = 19, log = "xy")
abline(0,1, col = "red")
```



Together, these plots suggest, that `R = 20` is sufficient for this dataset.

Note that with `pFdr(res1)`, we assumed that the marginal distributions of the statistics are exchangeable (see also `?randRotation::pFdr`) and thus pooling of the rotated statistics can be used. By pooling rotated statistics, the number of random rotations can be substantially reduced.

8 Correlation matrices with non-block design

Function `initBatchRandrot` implicitly assumes a block design of the sample correlation matrix and the restricted rotation matrix (see also `?randRotation::initBatchRandrot`). This means that correlations between samples are allowed within batches, but are zero between batches. Simply put, biological replicates or technical replicates (or any other cause of non-zero sample correlation) are contained within single batches and are not distributed to different batches. In this case, each batch has his own sample correlation matrix and correlation coefficients between batches are assumed to be zero. This assumption seems restrictive at first view, but is computationally efficient, as the random rotation can be performed for each batch independently. This is how `initBatchRandrot` is implemented. However, a general correlation matrix with non-block design (non-zero sample correlations between batches) can be initialised with `initRandrot`. Thus, `initBatchRandrot` simply provides a comfortable wrapper for sample correlation matrices with block design or for rotation of data with batch structure. For a correlation matrix of $I_{n\times n}$, `initRandrot` and `initBatchRandrot` are practically equivalent.

9 Batch effect correction with linear mixed models

9.1 Sample dataset

We now assume to have a dataset of repeated measures. We assume to have taken biopsies of 15 individuals. From each individual we have taken 1 biopsy of healthy control tissue and 1 biopsy of cancer tissue. This is a possible application for mixed models with the covariate "individual" as random effect. The hypothetic dataset was generated in 3 batches.

```
pdata$individual <- sort(c(1:15, 1:15))
colnames(pdata)[2] <- "tissue"
pdata$tissue <- c("Control", "Cancer")
pdata
#>    batch  tissue individual
#> 1      1 Control          1
#> 2      1  Cancer          1
#> 3      1 Control          2
#> 4      1  Cancer          2
#> 5      1 Control          3
#> 6      1  Cancer          3
#> 7      1 Control          4
#> 8      1  Cancer          4
#> 9      1 Control          5
#> 10     1  Cancer          5
#> 11     2 Control          6
#> 12     2  Cancer          6
#> 13     2 Control          7
#> 14     2  Cancer          7
#> 15     2 Control          8
```

```
#> 16     2  Cancer         8
#> 17     2 Control         9
#> 18     2  Cancer         9
#> 19     2 Control        10
#> 20     2  Cancer        10
#> 21     3 Control        11
#> 22     3  Cancer        11
#> 23     3 Control        12
#> 24     3  Cancer        12
#> 25     3 Control        13
#> 26     3  Cancer        13
#> 27     3 Control        14
#> 28     3  Cancer        14
#> 29     3 Control        15
#> 30     3  Cancer        15
```

As sample dataset, we take random normally distributed data with a random normally distributed individual effect (both with variance 1).

```
edata[,] <- rnorm(length(edata))
for(i in seq(1,ncol(edata),2)){
  tmp1 <- rnorm(nrow(edata))
  edata[,i] <- edata[,i] + tmp1
  edata[,i+1] <- edata[,i+1] + tmp1
}
```

9.2 Estimation of `cormat`

For random rotation of the dataset, we need an estimate of the correlation matrix `cormat` between sample replicates (of course different approaches than the following are possible for estimating `cormat`). As the data is not batch effect corrected, we estimate the correlation matrix for each batch separately and then average over all features and batches.

```
library(nlme)
df1 <- data.frame(pdata, d1 = edata[1,])
spl1 <- split(1:nrow(pdata), pdata$batch)

covs1 <- function(., df1, i){
  df1$d1 <- .

  me1 <- lme(d1 ~ tissue, data = df1[i,], random = ~1|individual)
  getVarCov(me1, type = "marginal")[[1]]
}

covs1 <- sapply(spl1,
                function(samps)rowMeans(apply(edata, 1, covs1, df1, samps)))
cov1 <- matrix(rowMeans(covs1), 2, 2)
cormat <- cov2cor(cov1)
cormat
#>        [,1]   [,2]
#> [1,] 1.0000 0.5266
```

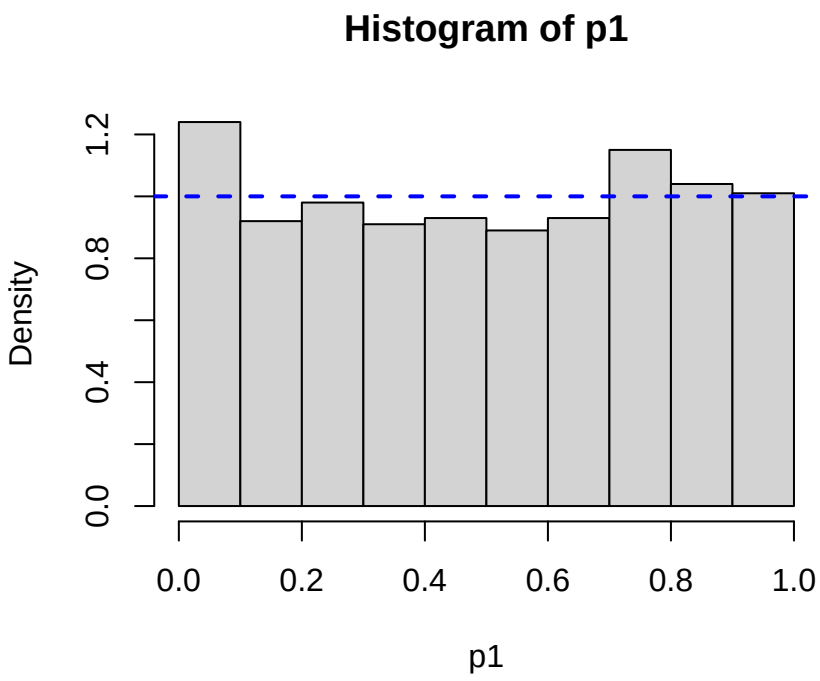
```
#> [2,] 0.5266 1.0000
```

As expected, the sample correlation is roughly 0.5, as the residual variance and the `individual` variance are both 1 in our sample dataset.

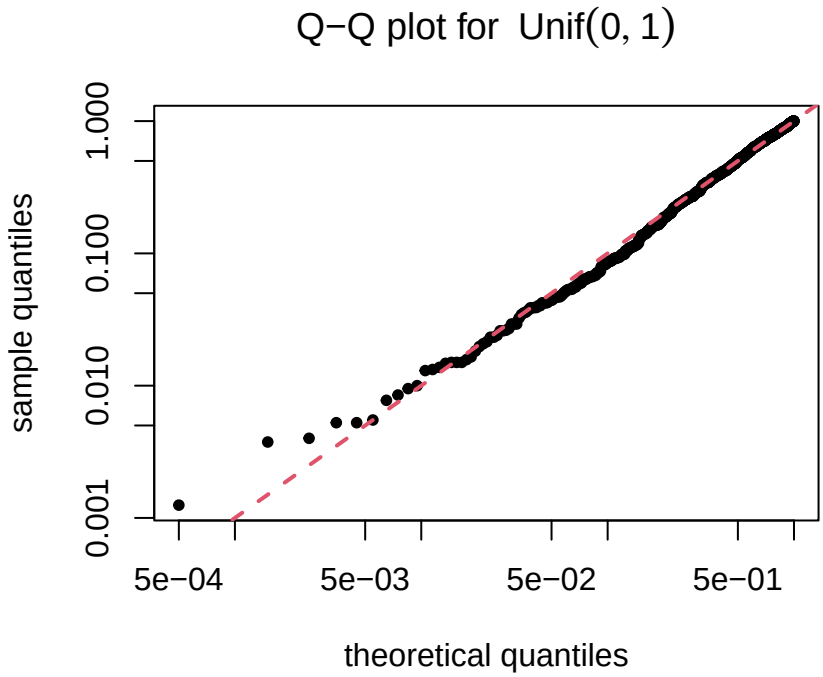
9.3 Random rotation

We can now initialise our random rotation object with `initBatchRandrot`and perform random rotation of our statistic of interest with `rotateStat`. We choose the absolute value of the t-statistic of coefficient `tissue` as statistic. We use the function `removeBatchEffect` from package *limma* for batch effect correction. Note that `removeBatchEffect` here is just a placeholder for any "black box batch effect correction procedure".

```
cormat <-  diag(5) %x% cormat
cormat <- list(cormat, cormat, cormat)

mod1 <- model.matrix(~1+tissue, pdata)
rr1 <- initBatchRandrot(Y = edata, X = mod1, coef.h = 2, batch = pdata$batch,
                        cormat = cormat)
#> Initialising batch "1"
#> Initialising batch "2"
#> Initialising batch "3"

statistic <- function(Y, batch, mod, df1){
  # Batch effect correction
  Y <- limma::removeBatchEffect(Y, batch = batch, design = mod)

  apply(Y, 1, function(j){
    df1$d1 <- j
    me0 <- nlme::lme(d1 ~ 1, data = df1, random = ~1|individual, method = "ML")
    me1 <- nlme::lme(d1 ~ tissue, data = df1, random = ~1|individual, method = "ML")

    abs(coef(me1)[1,2] / (sqrt(vcov(me1)[2,2])))
  })

}

rs1 <- rotateStat(initialised.obj = rr1, R = 4, statistic = statistic,
                  batch = pdata$batch, mod = mod1, df1 = df1, parallel = TRUE)

p1 <- pFdr(rs1)

hist(p1, freq = FALSE); abline(h = 1, lty = 2, lwd = 2, col = "blue")
```




```
qqunif(p1)
```



As expected, the p-value is roughly uniformly distributed.

10 Session info

```
sessionInfo()
#> R version 4.3.1 Patched (2023-06-17 r84564)
#> Platform: x86_64-apple-darwin20 (64-bit)
#> Running under: macOS Monterey 12.6.5
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRlapack.dylib;  LAPACK versio
#>
#> locale:
#> [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
#> time zone: America/New_York
#> tzcode source: internal
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods   base
#>
#> other attached packages:
#> [1] sva_3.50.0          BiocParallel_1.36.0 genefilter_1.84.0
#> [4] mgcv_1.9-0          nlme_3.1-163        limma_3.58.0
#> [7] randRotation_1.14.0 BiocStyle_2.30.0
#>
#> loaded via a namespace (and not attached):
#>  [1] bitops_1.0-7            lattice_0.22-5          RSQLite_2.3.1
#>  [4] digest_0.6.33           grid_4.3.1              evaluate_0.22
#>  [7] bookdown_0.36           fastmap_1.1.1           blob_1.2.4
#> [10] Matrix_1.6-1.1          AnnotationDbi_1.64.0    GenomeInfoDb_1.38.0
#> [13] DBI_1.1.3               survival_3.5-7          BiocManager_1.30.22
#> [16] httr_1.4.7              XML_3.99-0.14           Biostrings_2.70.0
#> [19] codetools_0.2-19        Rdpack_2.5              cli_3.6.1
#> [22] rlang_1.1.1             crayon_1.5.2            rbibutils_2.2.15
#> [25] XVector_0.42.0          Biobase_2.62.0          splines_4.3.1
#> [28] bit64_4.0.5             cachem_1.0.8            yaml_2.3.7
#> [31] tools_4.3.1             parallel_4.3.1          memoise_2.0.1
#> [34] annotate_1.80.0         locfit_1.5-9.8          GenomeInfoDbData_1.2.11
#> [37] BiocGenerics_0.48.0     vctrs_0.6.4             R6_2.5.1
#> [40] png_0.1-8               matrixStats_1.0.0       stats4_4.3.1
#> [43] zlibbioc_1.48.0         KEGGREST_1.42.0         edgeR_4.0.0
#> [46] S4Vectors_0.40.0        IRanges_2.36.0          bit_4.0.5
#> [49] Rcpp_1.0.11             statmod_1.5.0           xfun_0.40
#> [52] MatrixGenerics_1.14.0   knitr_1.44              xtable_1.8-4
#> [55] htmltools_0.5.6.1       rmarkdown_2.25          compiler_4.3.1
#> [58] RCurl_1.98-1.12
```

References

Hettegger, Peter, Klemens Vierlinger, and Andreas Weinhaeusel. 2021. "Random rotation for identifying differentially expressed genes with linear models following batch effect correction." *Bioinformatics*. https://doi.org/10.1093/bioinformatics/btab063.

Langsrud, Oyvind. 2005. "Rotation tests." *Statistics and Computing* 15 (1): 53–60. https://doi.org/10.1007/s11222-005-4789-5.

Nygaard, Vegard, Einar Andreas Rødland, and Eivind Hovig. 2015. "Methods that remove batch effects while retaining group differences may lead to exaggerated confidence in downstream analyses." *Biostatistics (Oxford, England)*, no. April 2016: kxv027. https://doi.org/10.1093/biostatistics/kxv027.

Yekutieli, Daniel, and Yoav Benjamini. 1999. "Resampling-based false discovery rate controlling multiple test procedures for correlated test statistics." *Journal of Statistical Planning and Inference* 82 (1-2): 171–96. https://doi.org/10.1016/S0378-3758(99)00041-5.