FRASER: Find RAre Splicing Events in RNA-seq

Christian Mertes[1]***, Ines Scheller***[1]***, Julien Gagneur***[1]

[1] Technische Universität München, Department of Informatics, Garching, Germany

October 26, 2021

Abstract

Genetic variants affecting splicing are a major cause of rare diseases yet their identification remains challenging. Recently, detecting splicing defects by RNA sequencing (RNA-seq) has proven to be an effective complementary avenue to genomic variant interpretation. However, no specialized method exists for the detection of aberrant splicing events in RNA-seq data. Here, we addressed this issue by developing the statistical method *FRASER* (Find RAre Splicing Events in RNA-seq). *FRASER* detects splice sites de novo, assesses both alternative splicing and intron retention, automatically controls for latent confounders using a denoising autoencoder, and provides significance estimates using an over-dispersed count fraction distribution. *FRASER* outperforms state-of-the-art approaches on simulated data and on enrichments for rare near-splice site variants in 48 tissues of the GTEx dataset. Application to a previously analysed rare disease dataset led to a new diagnostic by reprioritizing an aberrant exon truncation in TAZ. Altogether, we foresee *FRASER* as an important tool for RNA-seq based diagnostics of rare diseases.

If you use *FRASER* in published research, please cite:

Mertes C, Scheller I, Yepez V, *et al.* **Detection of aberrant splicing events in RNA-seq data with FRASER**, bioRxiv, 2019, *https://doi.org/10.1101/2019.12.18.866830*

Package

FRASER 1.6.0

Contents

1 Introduction

FRASER (Find RAre Splicing Evens in RNA-seq) is a tool for finding aberrant splicing events in RNA-seq samples. It works on the splice metrics ψ_5, ψ_3 and θ to be able to detect any type of aberrant splicing event from exon skipping over alternative donor usage to intron retention. To detect these aberrant events, *FRASER* uses a similar approach as the *OUTRIDER* package that aims to find aberrantly expressed genes and makes use of an autoencoder to automatically control for confounders within the data. *FRASER* also uses this autoencoder approach and models the read count ratios in the ψ values by fitting a beta binomial model to the ψ values obtained from RNA-seq read counts and correcting for apparent co-variations across samples. Similarly as in *OUTRIDER*, read counts that significantly deviate from the distribution are detected as outliers. A scheme of this approach is given in Figure 1.

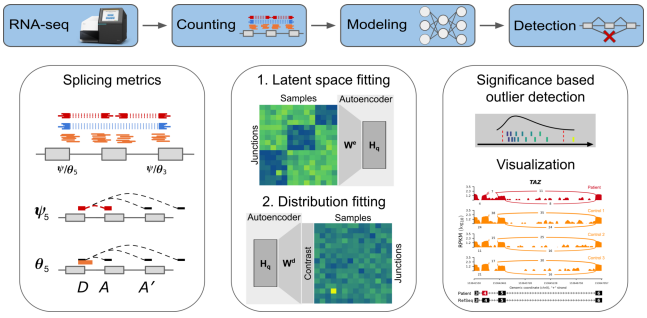


Figure 1: The *FRASER* splicing outlier detection workflow
The workflow starts with RNA-seq aligned reads and performs splicing outlier detection in three steps. First (left column), a splice site map is generated in an annotation-free fashion based on RNA-seq split reads. Split reads supporting exon-exon junctions as well as non-split reads overlapping splice sites are counted. Splicing metrics quantifying alternative acceptors (ψ_5), alternative donors (ψ_3) and splicing efficiencies at donors (θ_5) and acceptors (θ_3) are computed. Second (middle column), a statistical model is fitted for each splicing metric that controls for sample covariations (latent space fitting using a denoising autoencoder) and overdispersed count ratios (beta-binomial distribution). Third (right column), outliers are detected as data points significantly deviating from the fitted models. Candidates are then visualized with a genome browser.

FRASER uses the following splicing metrics as described by Pervouchine et al[1]: we compute for each sample, for donor D (5' splice site) and acceptor A (3' splice site) the ψ_5 and ψ_3 values, respectively, as:

$$\psi_5(D,A) = \frac{n(D,A)}{\sum_{A'} n(D,A')} \quad \boxed{1}$$

and

$$\psi_3(D,A) = \frac{n(D,A)}{\sum_{D'} n(D',A)}, \quad \boxed{2}$$

where $n(D,A)$ denotes the number of split reads spanning the intron between donor D and acceptor A and the summands in the denominators are computed over all acceptors found to splice with the donor of interest (Equation 1), and all donors

found to splice with the acceptor of interest (Equation 2). To not only detect alternative splicing but also partial or full intron retention, we also consider θ as a splicing efficiency metric.

$$\theta_5(D) = \frac{\sum_{A'} n(D, A')}{n(D) + \sum_{A'} n(D, A')} \quad (3)$$

and

$$\theta_3(A) = \frac{\sum_{D'} n(D', A)}{n(A) + \sum_{D'} n(D', A)}, \quad (4)$$

where $n(D)$ is the number of non-split reads spanning exon-intron boundary of donor D, and $n(A)$ is defined as the number of non-split reads spanning the intron-exon boundary of acceptor A. While we calculate θ for the 5' and 3' splice site separately, we do not distinguish later in the modeling step between θ_5 and θ_3 and hence call it jointly θ in the following.

2 Quick guide to *FRASER*

Here we quickly show how to do an analysis with *FRASER*, starting from a sample annotation table and the corresponding bam files. First, we create an *FraserDataSet* from the sample annotation and count the relevant reads in the bam files. Then, we compute the ψ/θ values and filter out introns that are just noise. Secondly, we run the full pipeline using the command `FRASER`. In the last step, we extract the results table from the *FraserDataSet* using the `results` function. Additionally, the user can create several analysis plots directly from the fitted *FraserDataSet* object. These plotting functions are described in section 4.4.

```
# load FRASER library
library(FRASER)

# count data
fds <- createTestFraserSettings()
fds <- countRNAData(fds)
fds
```

```
## ------------------- Sample data table ----------------
## # A tibble: 3 x 6
##   sampleID bamFile                    condition gene  pairedEnd SeqLevelStyle
##   <chr>    <chr>                          <int> <chr> <lgl>     <chr>
## 1 sample1  /tmp/RtmpfaJG3w/Rinst3d6d~         1 TIMM~ TRUE      UCSC
## 2 sample2  /tmp/RtmpfaJG3w/Rinst3d6d~         3 CLPP  TRUE      UCSC
## 3 sample3  /tmp/RtmpfaJG3w/Rinst3d6d~         2 MCOL~ TRUE      UCSC
##
## Number of samples:      3
## Number of junctions:    60
```

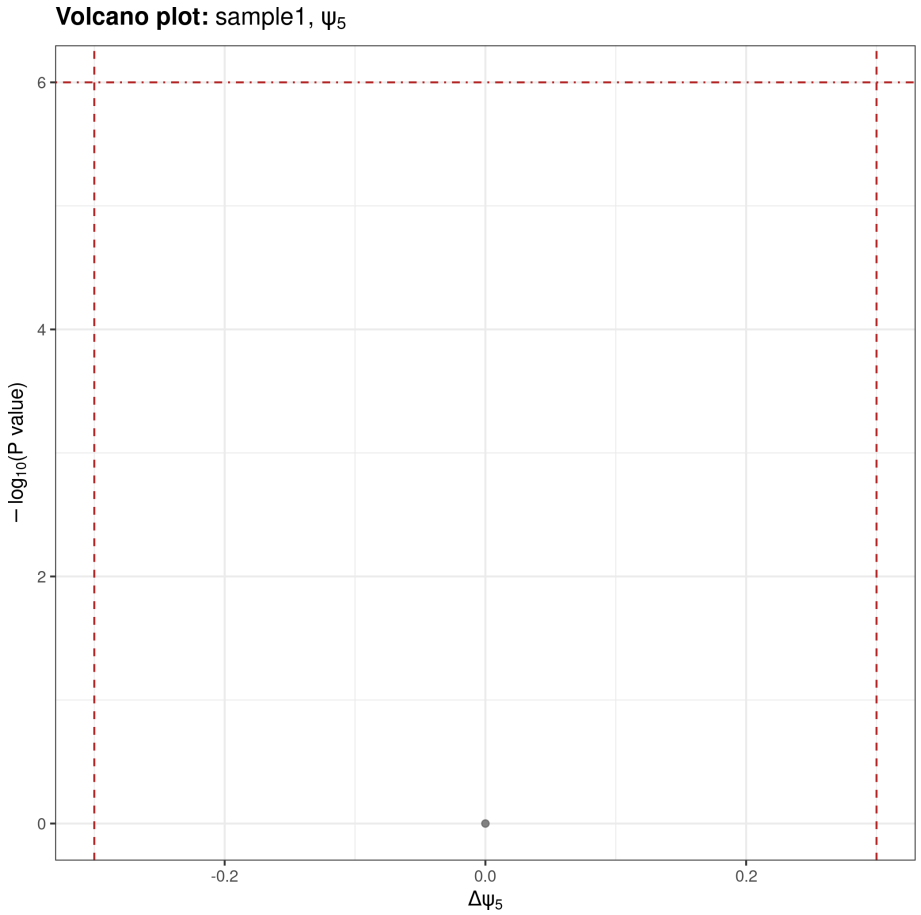
```
## Number of splice sites: 38
## assays(2): rawCountsJ rawCountsSS
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## ------------------- BAM parameters -------------------
## Default used with: bamMapqFilter=0
```

```
# compute stats
fds <- calculatePSIValues(fds)

# filtering junction with low expression
fds <- filterExpressionAndVariability(fds, minExpressionInOneSample=20,
       minDeltaPsi=0.0, filter=TRUE)

# fit the splicing model for each metric
# with a specific latentsapce dimension
fds <- FRASER(fds, q=c(psi5=2, psi3=3, theta=3))

# we provide two ways to anntoate introns with the corresponding gene symbols:
# the first way uses TxDb-objects provided by the user as shown here
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(org.Hs.eg.db)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
orgDb <- org.Hs.eg.db
fds <- annotateRangesWithTxDb(fds, txdb=txdb, orgDb=orgDb)

# alternatively, we also provide a way to use biomart for the annotation:
# fds <- annotateRanges(fds)

# get results: we recommend to use an FDR cutoff 0.05, but due to the small
# dataset size we extract all events and their associated values
# eg: res <- results(fds, zScoreCutoff=NA, padjCutoff=0.05, deltaPsiCutoff=0.3)
res <- results(fds, zScoreCutoff=NA, padjCutoff=NA, deltaPsiCutoff=NA)
res
```

```
## GRanges object with 226 ranges and 13 metadata columns:
##         seqnames              ranges strand | sampleID hgncSymbol
##            <Rle>           <IRanges>  <Rle> |    <Rle>      <Rle>
##     [1]     chr3 119242452-119242453      * |  sample2    TIMMDC1
##     [2]     chr3 119236163-119242452      * |  sample2    TIMMDC1
##     [3]     chr3 119236163-119242452      * |  sample3    TIMMDC1
##     [4]     chr3 119236162-119236163      * |  sample3    TIMMDC1
```

```
##     [5]     chr3 119242452-119242453      * |  sample3    TIMMDC1
##     ...      ...                  ...    ... .      ...        ...
##   [222]     chr3 119232567-119236051      * |  sample3    TIMMDC1
##   [223]     chr3 119217774-119217775      * |  sample1    TIMMDC1
##   [224]     chr3 119219541-119219542      * |  sample1    TIMMDC1
##   [225]    chr19     7592514-7592515      * |  sample3     MCOLN1
##   [226]    chr19     7592749-7592750      * |  sample3     MCOLN1
##         addHgncSymbols  type    pValue   padjust zScore psiValue deltaPsi
##                  <Rle> <Rle> <numeric> <numeric>  <Rle>    <Rle>    <Rle>
##     [1]               theta   0.73802         1    NaN        0        0
##     [2]                psi5   0.73804         1    NaN        1        0
##     [3]                psi5   0.73812         1    NaN        1        0
##     [4]               theta   0.73812         1   0.58        0        0
##     [5]               theta   0.73812         1    NaN        0        0
##     ...             ...   ...       ...       ...    ...      ...      ...
##   [222]                psi5         1         1  -0.58        1        0
##   [223]               theta         1         1   -0.7     0.33        0
##   [224]               theta         1         1  -1.15     0.18        0
##   [225]               theta         1         1   0.76      0.7        0
##   [226]               theta         1         1    NaN     0.73        0
##         meanCounts meanTotalCounts counts totalCounts
##              <Rle>           <Rle>  <Rle>       <Rle>
##     [1]          0          335.67      0         490
##     [2]     333.67          333.67    485         485
##     [3]     333.67          333.67    468         468
##     [4]          1          334.67      0         468
##     [5]          0          335.67      0         469
##     ...        ...             ...    ...         ...
##   [222]        274          285.33    430         432
##   [223]       8.33             269      8          24
##   [224]       4.67             269      4          22
##   [225]       3.67              49      7          10
##   [226]       3.33           49.67      8          11
##   -------
##   seqinfo: 2 sequences from an unspecified genome
```

```
# result visualization
plotVolcano(fds, sampleID="sample1", type="psi5", aggregate=TRUE)
```



3 A detailed *FRASER* analysis

The analysis workflow of *FRASER* for detecting rare aberrant splicing events in RNA-seq data can be divided into the following steps:

1. Data import or Counting reads 3.1
2. Data preprocessing and QC 3.2
3. Correcting for confounders 4.1
4. Calculate P-values 4.2
5. Calculate Z-scores 4.3
6. Visualize the results 4.4

Step 3-5 are wrapped up in one function `FRASER`, but each step can be called individually and parametrizied. Either way, data preprocessing should be done before starting the analysis, so that samples failing quality measurements or introns stemming from background noise are discarded.

Detailed explanations of each step are given in the following subsections.

For this tutorial we will use the a small example dataset that is contained in the package.

3.1 Data preparation

3.1.1 Creating a *FraserDataSet* and Counting reads

To start a RNA-seq data analysis with *FRASER* some preparation steps are needed. The first step is the creation of a *FraserDataSet* which derives from a RangedSummarizedExperiment object. To create the *FraserDataSet*, sample annotation and two count matrices are needed: one containing counts for the splice junctions, i.e. the split read counts, and one containing the splice site counts, i.e. the counts of non split reads overlapping with the splice sites present in the splice junctions.

You can first create the *FraserDataSet* with only the sample annotation and subsequently count the reads as described in 3.1.1. For this, we need a table with basic informations which then can be transformed into a *FraserSettings* object. The minimum of information per sample is an unique sample name, the path to the aligned bam file. Additionally groups can be specified for the P-value calculations later. If a **NA** is assigned no P-values will be calculated. An example sample table is given within the package:

```
sampleTable <- fread(system.file(
    "extdata", "sampleTable.tsv", package="FRASER", mustWork=TRUE))
head(sampleTable)
```

```
##    sampleID                 bamFile group    gene pairedEnd
## 1:  sample1 extdata/bam/sample1.bam     1 TIMMDC1      TRUE
## 2:  sample2 extdata/bam/sample2.bam     3    CLPP      TRUE
## 3:  sample3 extdata/bam/sample3.bam     2  MCOLN1      TRUE
```

To create a settings object for *FRASER* the constructor `FraserSettings` should be called with at least a sampleData table. For an example have a look into the `createTestFraserSettings`. In addition to the sampleData you can specify further parameters.

1. The parallel backend (a *BiocParallelParam* object)
2. The read filtering (a *ScanBamParam* object)
3. An output folder for the resulting figures and the cache
4. If the data is strand specific or not

The following shows how to create a example *FraserDataSet* with only the settings options from the sample annotation above:

```
# convert it to a bamFile list
bamFiles <- system.file(sampleTable[,bamFile], package="FRASER", mustWork=TRUE)
sampleTable[,bamFile:=bamFiles]
```

```
# create FRASER object
settings <- FraserDataSet(colData=sampleTable,
        workingDir=tempdir())

# show the FraserSettings object
settings
```

```
## ------------------- Sample data table ----------------
## # A tibble: 3 x 5
##   sampleID bamFile                                    group gene   pairedEnd
##   <chr>    <chr>                                      <int> <chr>  <lgl>
## 1 sample1  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FRASER~     1 TIMMD~ TRUE
## 2 sample2  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FRASER~     3 CLPP   TRUE
## 3 sample3  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FRASER~     2 MCOLN1 TRUE
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## -------------------- BAM parameters --------------------
## Default used with: bamMapqFilter=0
```

The *FraserDataSet* for this example data can also be generated through the function `createTestFraserSettings`:

```
settings <- createTestFraserSettings()
settings
```

```
## ------------------- Sample data table ----------------
## # A tibble: 3 x 5
##   sampleID bamFile                                  condition gene   pairedEnd
##   <chr>    <chr>                                        <int> <chr>  <lgl>
## 1 sample1  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FR~         1 TIMMD~ TRUE
## 2 sample2  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FR~         3 CLPP   TRUE
## 3 sample3  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FR~         2 MCOLN1 TRUE
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## -------------------- BAM parameters --------------------
## Default used with: bamMapqFilter=0
```

Counting of the reads are straight forward and is done through the `countRNAData` function. The only required parameter is the FraserSettings object. First all split reads are extracted from each individual sample and cached if enabled. Then a dataset wide junction map is created (all visible junctions over all samples). After that for each sample the non-spliced reads at each given donor and acceptor site is counted. The resulting *FraserDataSet* object contains two *SummarizedExperiment* objects for each the junctions and the splice sites.

```
# example of how to use parallelization: use 10 cores or the maximal number of
# available cores if fewer than 10 are available and use Snow if on Windows
if(.Platform$OS.type == "unix") {
    register(MulticoreParam(workers=min(10, multicoreWorkers())))
} else {
    register(SnowParam(workers=min(10, multicoreWorkers())))
}
```

```
# count reads
fds <- countRNAData(settings)
fds
```

```
## ------------------- Sample data table ----------------
## # A tibble: 3 x 5
##   sampleID bamFile                                  condition gene   pairedEnd
##   <chr>    <chr>                                        <int> <chr>  <lgl>
## 1 sample1  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FR~         1 TIMMD~ TRUE
## 2 sample2  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FR~         3 CLPP   TRUE
## 3 sample3  /tmp/RtmpfaJG3w/Rinst3d6d6e40817e9c/FR~         2 MCOLN1 TRUE
##
## Number of samples:      3
## Number of junctions:    60
## Number of splice sites: 38
## assays(2): rawCountsJ rawCountsSS
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## ------------------- BAM parameters -------------------
## Default used with: bamMapqFilter=0
```

3.1.2 Creating a *FraserDataSet* from existing count matrices

If the count matrices already exist, you can use these matrices directly together with the sample annotation from above to create the *FraserDataSet*:

```
# example sample annoation for precalculated count matrices
sampleTable <- fread(system.file("extdata", "sampleTable_countTable.tsv",
        package="FRASER", mustWork=TRUE))
head(sampleTable)
```

```
##    sampleID                bamFile group    gene
## 1:  sample1 extdata/bam/sample1.bam     1 TIMMDC1
## 2:  sample2 extdata/bam/sample2.bam     1 TIMMDC1
## 3:  sample3 extdata/bam/sample3.bam     2  MCOLN1
## 4:  sample4 extdata/bam/sample4.bam     3    CLPP
## 5:  sample5 extdata/bam/sample5.bam    NA    NHDF
## 6:  sample6 extdata/bam/sample6.bam    NA    NHDF
```

```
# get raw counts
junctionCts   <- fread(system.file("extdata", "raw_junction_counts.tsv.gz",
        package="FRASER", mustWork=TRUE))
head(junctionCts)
```

```
##    seqnames   start     end  width strand sample1 sample2 sample3 sample4
## 1:    chr19 7126380 7690902 564523      *       0       1       0       0
## 2:    chr19 7413458 7615986 202529      *       0       1       0       0
## 3:    chr19 7436801 7703913 267113      *       0       0       0       0
## 4:    chr19 7466307 7607189 140883      *       0       0       0       0
## 5:    chr19 7471938 7607808 135871      *       1       0       0       0
## 6:    chr19 7479042 7625600 146559      *       0       0       0       0
##    sample5 sample6 sample7 sample8 sample9 sample10 sample11 sample12
## 1:       0       0       0       0       0        0        0        0
## 2:       0       0       0       0       0        0        0        0
## 3:       0       1       0       0       0        0        0        0
## 4:       0       0       1       0       0        0        0        0
## 5:       0       0       0       0       0        0        0        0
## 6:       0       0       0       0       0        0        0        1
##    startID endID
## 1:       1    90
## 2:       2    91
## 3:       3    92
## 4:       4    93
## 5:       5    94
## 6:       6    95
```

```
spliceSiteCts <- fread(system.file("extdata", "raw_site_counts.tsv.gz",
        package="FRASER", mustWork=TRUE))
head(spliceSiteCts)
```

```
##    seqnames   start     end width strand spliceSiteID  type sample1 sample2
## 1:    chr19 7126379 7126380     2      *            1 Donor       0       0
## 2:    chr19 7413457 7413458     2      *            2 Donor       0       0
```

```
## 3:    chr19 7436800 7436801     2      *            3 Donor       0       0
## 4:    chr19 7466306 7466307     2      *            4 Donor       0       0
## 5:    chr19 7471937 7471938     2      *            5 Donor       0       0
## 6:    chr19 7479041 7479042     2      *            6 Donor       0       0
##    sample3 sample4 sample5 sample6 sample7 sample8 sample9 sample10 sample11
## 1:       0       0       0       0       0       0       0        0        0
## 2:       0       0       0       0       0       0       0        0        0
## 3:       0       0       0       0       0       0       0        0        0
## 4:       0       0       0       0       0       0       0        0        0
## 5:       0       0       0       0       0       0       0        0        0
## 6:       0       0       0       0       0       0       0        0        0
##    sample12
## 1:        0
## 2:        0
## 3:        0
## 4:        0
## 5:        0
## 6:        0
```

```
# create FRASER object
fds <- FraserDataSet(colData=sampleTable, junctions=junctionCts,
        spliceSites=spliceSiteCts, workingDir=tempdir())
fds
```

```
## ------------------- Sample data table -----------------
## # A tibble: 12 x 4
##    sampleID bamFile                  group gene
##    <chr>    <chr>                    <int> <chr>
##  1 sample1  extdata/bam/sample1.bam      1 TIMMDC1
##  2 sample2  extdata/bam/sample2.bam      1 TIMMDC1
##  3 sample3  extdata/bam/sample3.bam      2 MCOLN1
##  4 sample4  extdata/bam/sample4.bam      3 CLPP
##  5 sample5  extdata/bam/sample5.bam     NA NHDF
##  6 sample6  extdata/bam/sample6.bam     NA NHDF
##  7 sample7  extdata/bam/sample7.bam     NA NHDF
##  8 sample8  extdata/bam/sample8.bam     NA NHDF
##  9 sample9  extdata/bam/sample9.bam     NA NHDF
## 10 sample10 extdata/bam/sample10.bam    NA NHDF
## 11 sample11 extdata/bam/sample11.bam    NA NHDF
## 12 sample12 extdata/bam/sample12.bam    NA NHDF
##
## Number of samples:      12
## Number of junctions:    123
## Number of splice sites: 165
## assays(2): rawCountsJ rawCountsSS
##
```

```
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## ------------------- BAM parameters -------------------
## Default used with: bamMapqFilter=0
```

3.2 Data preprocessing and QC

As with gene expression analysis, a good quality control of the raw data is crucial. For some hints please refere to our workshop slides[1].

[1] http://tinyurl.com/RNA-ASHG-presentation

At the time of writing this vignette, we recommend that the RNA-seq data should be aligned with a splice-aware aligner like STAR[2] or GEM[3]. To gain better results, at least 20 samples should be sequenced and they should be processed with the same protocol and origin from the same tissue.

3.2.1 Filtering

Before we can filter the data, we have to compute the main splicing metric: the ψ-value (Percent Spliced In).

```
fds <- calculatePSIValues(fds)
fds
```

```
## ------------------- Sample data table -----------------
## # A tibble: 12 x 4
##    sampleID bamFile                  group gene
##    <chr>    <chr>                    <int> <chr>
##  1 sample1  extdata/bam/sample1.bam      1 TIMMDC1
##  2 sample2  extdata/bam/sample2.bam      1 TIMMDC1
##  3 sample3  extdata/bam/sample3.bam      2 MCOLN1
##  4 sample4  extdata/bam/sample4.bam      3 CLPP
##  5 sample5  extdata/bam/sample5.bam     NA NHDF
##  6 sample6  extdata/bam/sample6.bam     NA NHDF
##  7 sample7  extdata/bam/sample7.bam     NA NHDF
##  8 sample8  extdata/bam/sample8.bam     NA NHDF
##  9 sample9  extdata/bam/sample9.bam     NA NHDF
## 10 sample10 extdata/bam/sample10.bam    NA NHDF
## 11 sample11 extdata/bam/sample11.bam    NA NHDF
## 12 sample12 extdata/bam/sample12.bam    NA NHDF
##
## Number of samples:      12
## Number of junctions:    123
## Number of splice sites: 165
```

```
## assays(11): rawCountsJ psi5 ... rawOtherCounts_theta delta_theta
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## -------------------- BAM parameters --------------------
## Default used with: bamMapqFilter=0
```

Now we can have some cut-offs to filter down the number of junctions we want to test later on.

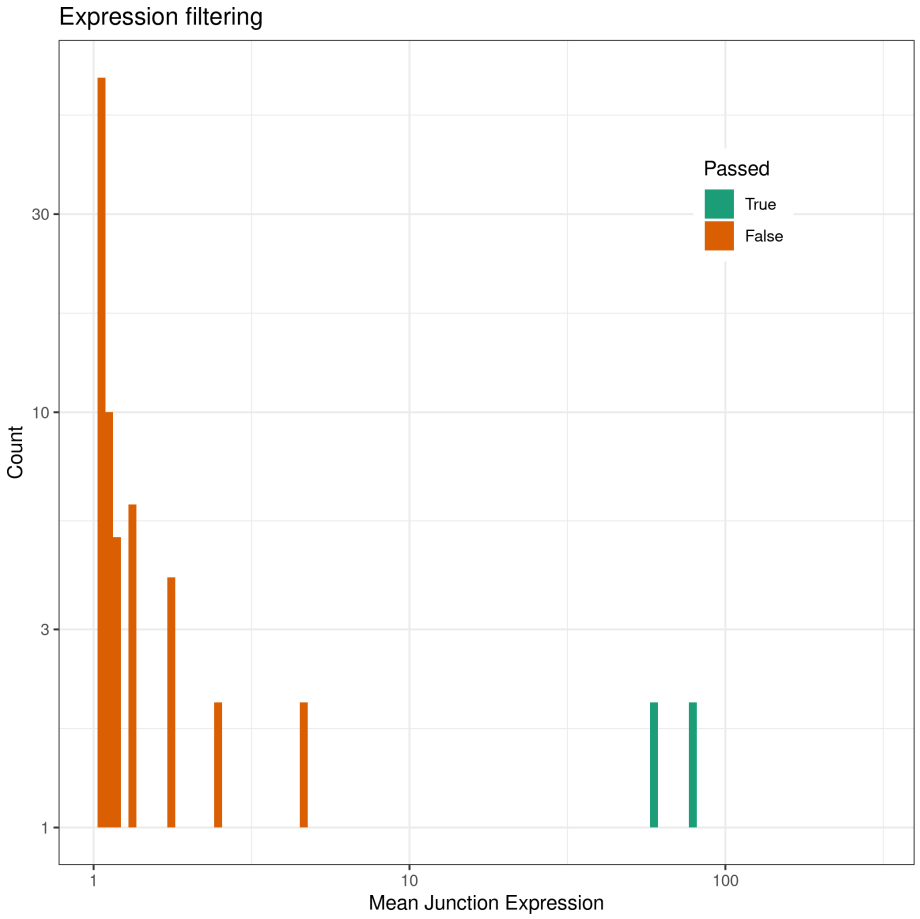
Currently, we keep only junctions which support the following:

- At least one sample has 20 reads
- 5% of the samples have at least 1 read

Furthemore one could filter for:

- At least one sample has a $|\Delta\psi|$ of 0.1

```
fds <- filterExpressionAndVariability(fds, minDeltaPsi=0.0, filter=FALSE)

plotFilterExpression(fds, bins=100)
```



After looking at the expression distribution between filtered and unfiltered junctions, we can now subset the dataset:

```
fds_filtered <- fds[mcols(fds, type="j")[,"passed"],]
fds_filtered
```

```
## ------------------- Sample data table ----------------
## # A tibble: 12 x 4
##    sampleID bamFile                 group gene
##    <chr>    <chr>                   <int> <chr>
##  1 sample1  extdata/bam/sample1.bam     1 TIMMDC1
##  2 sample2  extdata/bam/sample2.bam     1 TIMMDC1
##  3 sample3  extdata/bam/sample3.bam     2 MCOLN1
##  4 sample4  extdata/bam/sample4.bam     3 CLPP
##  5 sample5  extdata/bam/sample5.bam    NA NHDF
##  6 sample6  extdata/bam/sample6.bam    NA NHDF
##  7 sample7  extdata/bam/sample7.bam    NA NHDF
##  8 sample8  extdata/bam/sample8.bam    NA NHDF
##  9 sample9  extdata/bam/sample9.bam    NA NHDF
## 10 sample10 extdata/bam/sample10.bam   NA NHDF
```

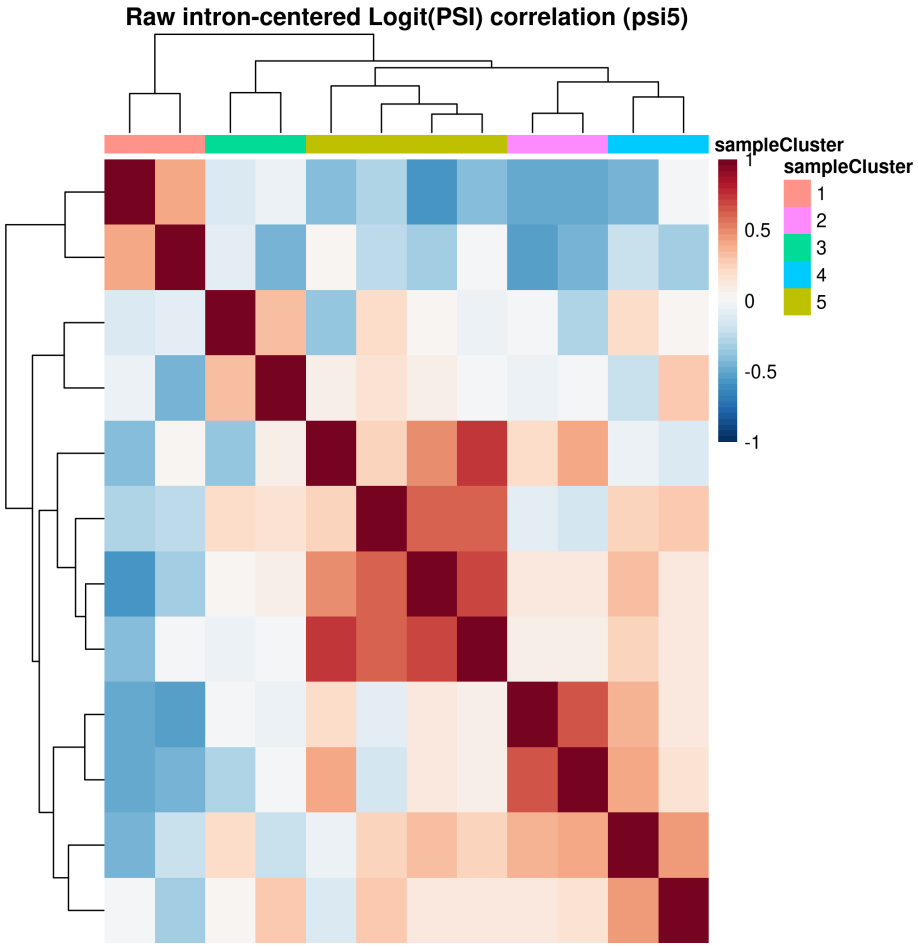
```
## 11 sample11 extdata/bam/sample11.bam    NA NHDF
## 12 sample12 extdata/bam/sample12.bam    NA NHDF
##
## Number of samples:      12
## Number of junctions:    20
## Number of splice sites: 38
## assays(11): rawCountsJ psi5 ... rawOtherCounts_theta delta_theta
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## -------------------- BAM parameters --------------------
## Default used with: bamMapqFilter=0

# filtered_fds not further used for this tutorial because the example dataset
# is otherwise too small
```

3.2.2 Sample co-variation

Since ψ values are ratios within a sample, one might think that there should not be as much correlation structure as observed in gene expression data within the splicing data.

This is not true as we do see strong sample co-variation across different tissues and cohorts. Let's have a look into our data to see if we do have correlation structure or not. To have a better estimate, we use the logit transformed ψ values to compute the correlation.

```
# Heatmap of the sample correlation
plotCountCorHeatmap(fds, type="psi5", logit=TRUE, normalized=FALSE)
```



It is also possible to visualize the correlation structure of the logit transformed ψ values of the $topJ$ most variable introns for all samples:

```
# Heatmap of the intron/sample expression
plotCountCorHeatmap(fds, type="psi5", logit=TRUE, normalized=FALSE,
    plotType="junctionSample", topJ=100, minDeltaPsi = 0.01)
```

3.3 Detection of aberrant splicing events

After preprocessing the raw data and visualizing it, we can start our analysis. Let's start with the first step in the aberrant splicing detection: the model fitting.

3.3.1 Fitting the splicing model

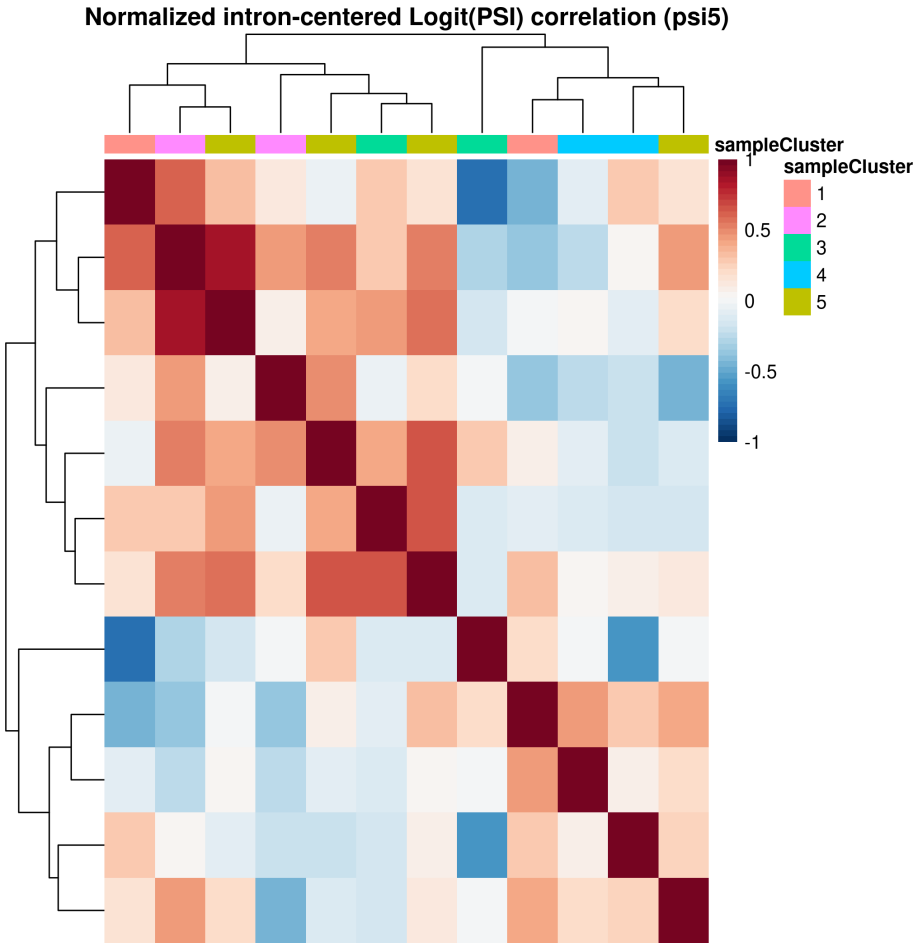
During the fitting procedure, we will normalize the data and correct for confounding effects by using a denoising autoencoder. Here we use a predefined latent space with a dimension $q = 10$. Using the correct dimension is crucial to have the best

performance (see 4.1.1). Alternatively, one can also use a PCA to correct the data. The wrapper function `FRASER` both fits the model and calculates the p-values and z-scores for all ψ types. For more details see section 4.

```
# This is computational heavy on real size datasets and can take awhile
fds <- FRASER(fds, q=c(psi5=3, psi3=5, theta=2))
```

To check whether the correction worked, we can have a look at the correlation heatmap using the normalized ψ values from the fit.

```
plotCountCorHeatmap(fds, type="psi5", normalized=TRUE, logit=TRUE)
```



3.3.2 Calling splicing outliers

Before we extract the results, we should add the human readable HGNC symbols. *FRASER* comes already with an annotation function. The function uses *biomaRt* in the background to overlap the genomic ranges with the known HGNC symbols. To have more flexibilty on the annotation, one can also provide a custom 'txdb' object to annotate the HGNC symbols.

Here we assume a beta binomial distribution and call outliers based on the significance level. The user can choose between a p value cutoff, a Z score cutoff or a cutoff on the $\Delta\psi$ values between the observed and expected ψ values or both.

```
# annotate introns with the HGNC symbols of the corresponding gene
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(org.Hs.eg.db)

txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
orgDb <- org.Hs.eg.db
fds <- annotateRangesWithTxDb(fds, txdb=txdb, orgDb=orgDb)
# fds <- annotateRanges(fds) # alternative way using biomaRt

# retrieve results with default and recommended cutoffs (padj <= 0.05 and
# |deltaPsi| >= 0.3)
res <- results(fds)
```

3.3.3 Interpreting the result table

The function `results` retrieves significant events based on the specified cutoffs as a *GRanges* object which contains the genomic location of the splice junction or splice site that was found as aberrant and the following additional information:

- sampleID: the sampleID in which this aberrant event occurred
- hgncSymbol: the gene symbol of the gene that contains the splice junction or site if available
- type: the metric for which the aberrant event was detected (either psi5 for ψ_5, psi3 for ψ_3 or theta for θ)
- pValue, padjust, zScore: the p-value, adjusted p-value and z-score of this event
- psiValue: the value of ψ_5, ψ_3 or θ metric (depending on the type column) of this junction or splice site for the sample in which it is detected as aberrant
- deltaPsi: the $\Delta\psi$-value of the event in this sample, which is the difference between the actual observed ψ and the expected ψ
- meanCounts: the mean count (k) of reads mapping to this splice junction or site over all samples
- meanTotalCounts: the mean total count (n) of reads mapping to the same donor or acceptor site as this junction or site over all samples
- counts, totalCounts: the count (k) and total count (n) of the splice junction or site for the sample where it is detected as aberrant

Please refer to section 1 for more information about the metrics ψ_5, ψ_3 and θ and their definition. In general, an aberrant ψ_5 value might indicate aberrant acceptor site usage of the junction where the event is detected; an aberrant ψ_3 value might

indicate aberrant donor site usage of the junction where the event is detected; and an aberrant θ value might indicate partial or full intron retention, or exon truncation or elongation. We recommend using a genome browser to investigate interesting detected events in more detail.

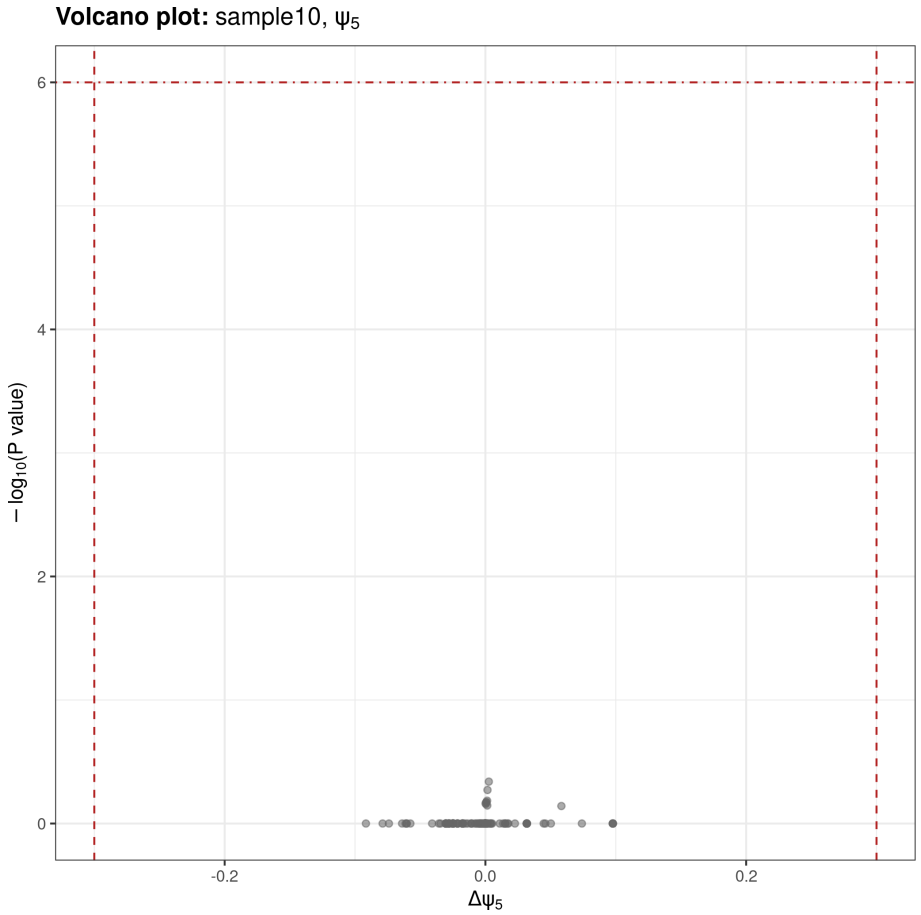
```
# to show result visualization functions for this tuturial, zScore cutoff used
res <- results(fds, zScoreCutoff=2, padjCutoff=NA, deltaPsiCutoff=0.1)
res
```

```
## GRanges object with 4 ranges and 13 metadata columns:
##       seqnames              ranges strand | sampleID hgncSymbol
##          <Rle>           <IRanges>  <Rle> |    <Rle>      <Rle>
##   119     chr3 119217435-119217436      * |  sample9    TIMMDC1
##    50    chr19     7593607-7593608      * | sample12     MCOLN1
##   125     chr3 119217780-119217781      * |  sample5    TIMMDC1
##    60    chr19     7595255-7595256      * |  sample5     MCOLN1
##       addHgncSymbols  type    pValue   padjust zScore psiValue deltaPsi
##                <Rle> <Rle> <numeric> <numeric>  <Rle>    <Rle>    <Rle>
##   119               theta  0.010509         1  -2.51     0.84    -0.11
##    50               theta  0.032500         1   2.36        1     0.38
##   125               theta  0.037823         1   2.15        1     0.16
##    60               theta  0.285720         1   -2.3     0.88    -0.12
##       meanCounts meanTotalCounts counts totalCounts
##            <Rle>           <Rle>  <Rle>       <Rle>
##   119      86.58           91.58     61          73
##    50       1.42            2.67      7           7
##   125       8.17           10.25     16          16
##    60      68.92           69.92     15          17
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

3.4 Finding splicing candidates in patients

Let's hava a look at sample 10 and check if we got some splicing candidates for this sample.

```
plotVolcano(fds, type="psi5", "sample10")
```



Which are the splicing events in detail?

```
sampleRes <- res[res$sampleID == "sample10"]
sampleRes
```

```
## GRanges object with 0 ranges and 13 metadata columns:
##    seqnames    ranges strand | sampleID hgncSymbol addHgncSymbols  type
##       <Rle> <IRanges>  <Rle> |    <Rle>      <Rle>          <Rle> <Rle>
##       pValue   padjust zScore psiValue deltaPsi meanCounts meanTotalCounts
##    <numeric> <numeric>  <Rle>    <Rle>    <Rle>      <Rle>           <Rle>
##    counts totalCounts
##     <Rle>       <Rle>
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

To have a closer look at the junction level, use the following functions:

```
plotExpression(fds, type="psi5", result=sampleRes[1])
plotExpectedVsObservedPsi(fds, result=sampleRes[1])
```

3.5 Saving and loading a *FraserDataSet*

A *FraserDataSet* object can be easily saved and reloaded at any time as follows:

```
# saving a fds
workingDir(fds) <- tempdir()
name(fds) <- "ExampleAnalysis"
saveFraserDataSet(fds, dir=workingDir(fds), name=name(fds))
```

```
## ------------------- Sample data table ----------------
## # A tibble: 12 x 4
##    sampleID bamFile                  group gene
##    <chr>    <chr>                    <int> <chr>
##  1 sample1  extdata/bam/sample1.bam      1 TIMMDC1
##  2 sample2  extdata/bam/sample2.bam      1 TIMMDC1
##  3 sample3  extdata/bam/sample3.bam      2 MCOLN1
##  4 sample4  extdata/bam/sample4.bam      3 CLPP
##  5 sample5  extdata/bam/sample5.bam     NA NHDF
##  6 sample6  extdata/bam/sample6.bam     NA NHDF
##  7 sample7  extdata/bam/sample7.bam     NA NHDF
##  8 sample8  extdata/bam/sample8.bam     NA NHDF
##  9 sample9  extdata/bam/sample9.bam     NA NHDF
## 10 sample10 extdata/bam/sample10.bam    NA NHDF
## 11 sample11 extdata/bam/sample11.bam    NA NHDF
## 12 sample12 extdata/bam/sample12.bam    NA NHDF
##
## Number of samples:      12
## Number of junctions:    123
## Number of splice sites: 165
## assays(26): rawCountsJ psi5 ... padjBetaBinomial_theta zScores_theta
##
## ---------------------- Settings ----------------------
## Analysis name:               ExampleAnalysis
## Analysis is strand specific: no
## Working directory:           '/tmp/Rtmp0NYa6W'
##
## ------------------- BAM parameters -------------------
## Default used with: bamMapqFilter=0
```

```
# two ways of loading a fds by either specifying the directory and anaysis name
# or directly giving the path the to fds-object.RDS file
fds <- loadFraserDataSet(dir=workingDir(fds), name=name(fds))
fds <- loadFraserDataSet(file=file.path(workingDir(fds),
    "savedObjects", "ExampleAnalysis", "fds-object.RDS"))
```

4 More details on *FRASER*

The function `FRASER` is a convenient wrapper function that takes care of correcting for confounders, fitting the beta binomial distribution and calculating p-values and z-scores for all ψ types. To have more control over the individual steps, the different functions can also be called separately. The following sections give a short explanation of these steps.

4.1 Correction for confounders

The wrapper function `FRASER` and the underlying function `fit` method offer different methods to automatically control for confounders in the data. Currently the following methods are implemented:

- AE: uses a beta-binomial AE
- PCA-BB-Decoder: uses a beta-binomial AE where PCA is used to find the latent space (encoder) due to speed reasons
- PCA: uses PCA for both the encoder and the decoder
- BB: no correction for confounders, fits a beta binomial distribution directly on the raw counts

```
# Using an alternative way to correct splicing ratios
# here: only 2 iteration to speed the calculation up
# for the vignette, the default is 15 iterations
fds <- fit(fds, q=3, type="psi5", implementation="PCA-BB-Decoder",
           iterations=2)
```

```
##
## TRUE
##  123
## [1] "Initial PCA loss: 1.74548927025931"
## [1] "Finished with fitting the E matrix. Starting now with the D fit. ..."
## [1] "Tue Oct 26 17:29:56 2021: Iteration: final_1 loss: 1.04552024657028 (mean); 4.0595912087300
## [1] "Tue Oct 26 17:29:57 2021: Iteration: final_2 loss: 1.01544950514076 (mean); 3.3238038197927
## [1] "2 Final betabin-AE loss: 1.01544950514076"
```

4.1.1 Finding the dimension of the latent space

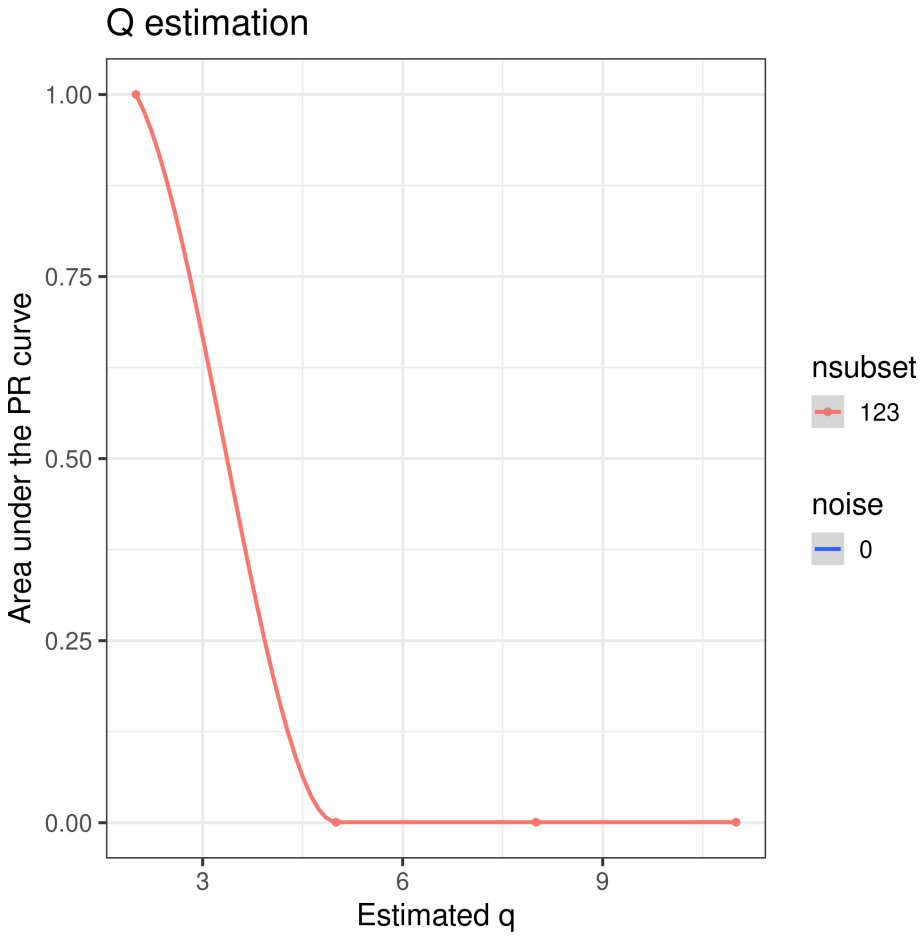
For the previous call, the dimension q of the latent space has been fixed to $q = 10$. Since working with the correct q is very important, the *FRASER* package also provides the function `optimHyperParams` that can be used to estimate the dimension q of the latent space of the data. It works by artificially injecting outliers into the data and then comparing the AUC of recalling these outliers for different values of q. Since this hyperparameter optimization step can take some time for the full dataset, we only show it here for a subset of the dataset:

```
set.seed(42)
# hyperparameter opimization
fds <- optimHyperParams(fds, type="psi5", plot=FALSE)

# retrieve the estimated optimal dimension of the latent space
bestQ(fds, type="psi5")
```

```
## [1] 2
```

The results from this hyper parameter optimization can be visualized with the function `plotEncDimSearch`.

```
plotEncDimSearch(fds, type="psi5")
```



4.2 P-value calculation

After determining the fit parameters, two-sided beta binomial P-values are computed using the following equation:

$$p_{ij} = 2 \cdot min \left\{ \frac{1}{2}, \sum_{0}^{k_{ij}} BB(k_{ij}, n_{ij}, \mu_{ij}, \rho_i), 1 - \sum_{0}^{k_{ij-1}} BB(k_{ij}, n_{ij}, \mu_{ij}, \rho_i) \right\}, \quad \boxed{5}$$

where the $\frac{1}{2}$ term handles the case of both terms exceeding 0.5, which can happen due to the discrete nature of counts. Here μ_{ij} are computed as the product of the fitted correction values from the autoencoder and the fitted mean adjustements.

```
fds <- calculatePvalues(fds, type="psi5")
head(pVals(fds, type="psi5"))
```

```
##   sample1 sample2 sample3 sample4 sample5 sample6 sample7 sample8 sample9
## 1       1       1       1       1       1       1       1       1       1
## 2       1       1       1       1       1       1       1       1       1
## 3       1       1       1       1       1       1       1       1       1
## 4       1       1       1       1       1       1       1       1       1
## 5       1       1       1       1       1       1       1       1       1
## 6       1       1       1       1       1       1       1       1       1
##   sample10 sample11 sample12
## 1        1        1        1
## 2        1        1        1
## 3        1        1        1
## 4        1        1        1
## 5        1        1        1
## 6        1        1        1
```

Afterwards, adjusted p-values can be calculated. Multiple testing correction is done across all junctions in a per-sample fashion using Benjamini-Yekutieli's false discovery rate method[4]. Alternatively, all adjustment methods supported by `p.adjust` can be used via the `method` argument.

```
fds <- calculatePadjValues(fds, type="psi5", method="BY")
head(padjVals(fds,type="psi5"))
```

```
##   sample1 sample2 sample3 sample4 sample5 sample6 sample7 sample8 sample9
## 1       1       1       1       1       1       1       1       1       1
## 2       1       1       1       1       1       1       1       1       1
## 3       1       1       1       1       1       1       1       1       1
## 4       1       1       1       1       1       1       1       1       1
## 5       1       1       1       1       1       1       1       1       1
## 6       1       1       1       1       1       1       1       1       1
```

```
##   sample10 sample11 sample12
## 1        1        1        1
## 2        1        1        1
## 3        1        1        1
## 4        1        1        1
## 5        1        1        1
## 6        1        1        1
```

4.3 Z-score calculation

To calculate z-scores on the logit transformed $\Delta\psi$ values and to store them in the *FraserDataSet* object, the function `calculateZScores` can be called. The Z-scores can be used for visualization, filtering, and ranking of samples. The Z-scores are calculated as follows:

$$z_{ij} = \frac{\delta_{ij} - \bar{\delta_j}}{sd(\delta_j)} \quad (6)$$

$$\delta_{ij} = logit(\frac{k_{ij}+1}{n_{ij}+2}) - logit(\mu_{ij}),$$

where δ_{ij} is the difference on the logit scale between the measured counts and the counts after correction for confounders and $\bar{\delta_j}$ is the mean of intron j.

```
fds <- calculateZscore(fds, type="psi5")
head(zScores(fds, type="psi5"))
```

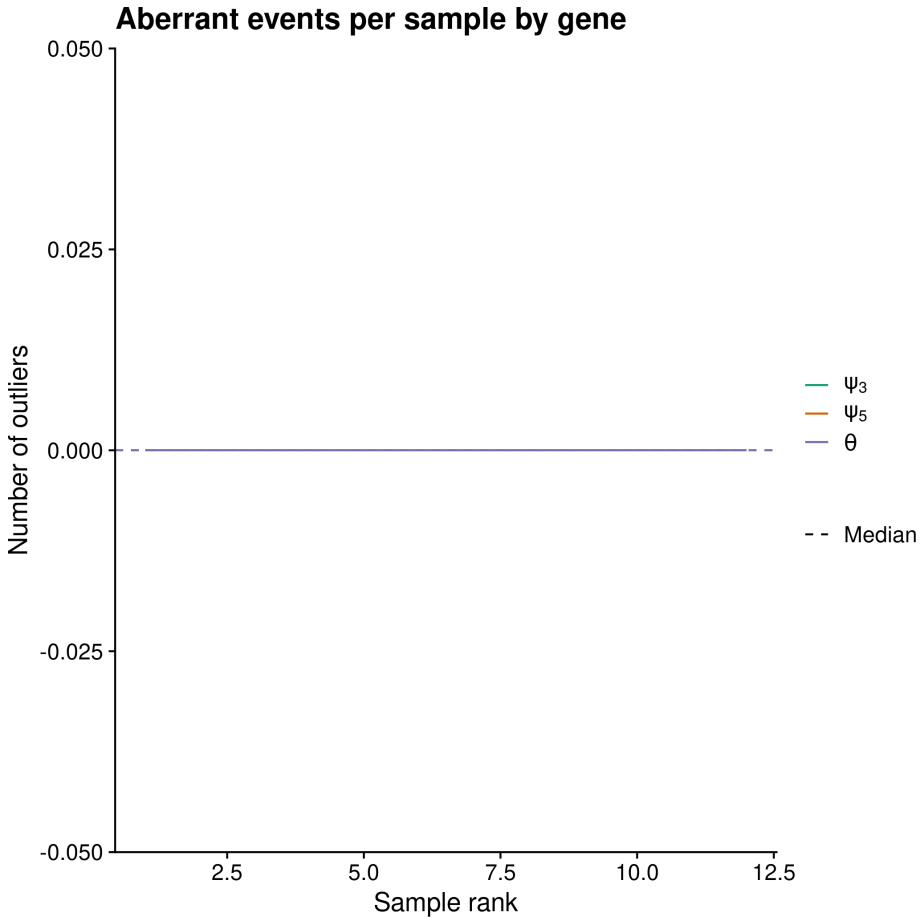
```
##      sample1    sample2     sample3     sample4     sample5     sample6
## 1 -2.2570915  2.3811586 -0.13635100  0.05653623 -0.01029320 -0.13318122
## 2 -2.2570631  2.3811887 -0.13644585  0.05671451 -0.01030313 -0.13317308
## 3 -0.1340479 -0.1276374 -0.29445741 -0.35544944 -0.31797016  3.16616030
## 4 -0.1550715  0.1576705 -0.14643680 -0.57021523 -0.24990096 -0.40097482
## 5  1.4111351 -2.6499939  0.07996799 -0.44844152 -0.01920583 -0.09238437
## 6  0.1535881  0.3966956 -0.21452236 -0.66819742 -0.36244648 -0.39814987
##      sample7     sample8    sample9   sample10    sample11   sample12
## 1 -0.1849994 -0.10910300 -0.1784178  0.2463657  0.11579225  0.2095844
## 2 -0.1849349 -0.10917413 -0.1784672  0.2462126  0.11591058  0.2095351
## 3 -0.3212759 -0.30113046 -0.3047008 -0.3189796 -0.35310240 -0.3374088
## 4  3.1066457 -0.19923297 -0.4117873 -0.1367544 -0.56061068 -0.4333315
## 5 -0.2806726  0.07451828  0.1586799  0.9960138 -0.04581502  0.8161983
## 6 -0.4846997 -0.26125498 -0.3319657 -0.2415962 -0.61782667  3.0303757
```

4.4 Result visualization

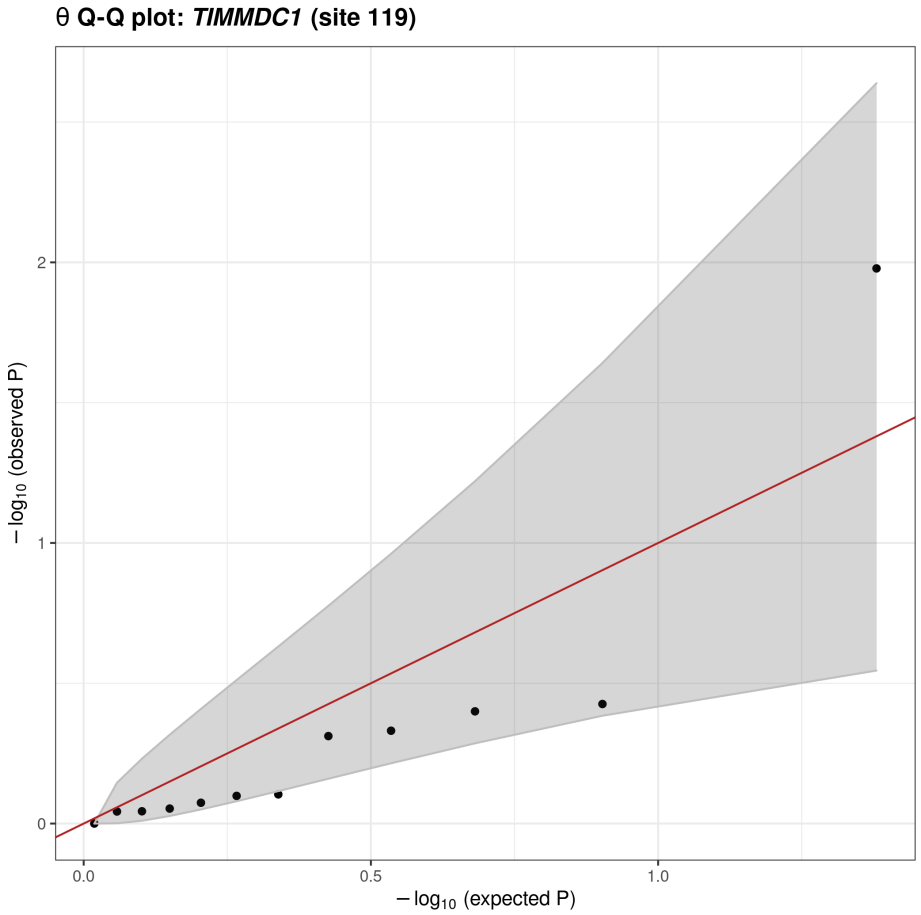
In addition to the plotting methods `plotVolcano`, `plotExpression`, `plotExpectedVsObservedPsi`, `plotFilterExpression` and `plotEncDimSearch` used above, the *FRASER* package provides two additional functions to visualize the results:

`plotAberrantPerSample` displays the number of aberrant events per sample based on the given cutoff values and `plotQQ` gives a quantile-quantile plot either for a single junction/splice site or globally.

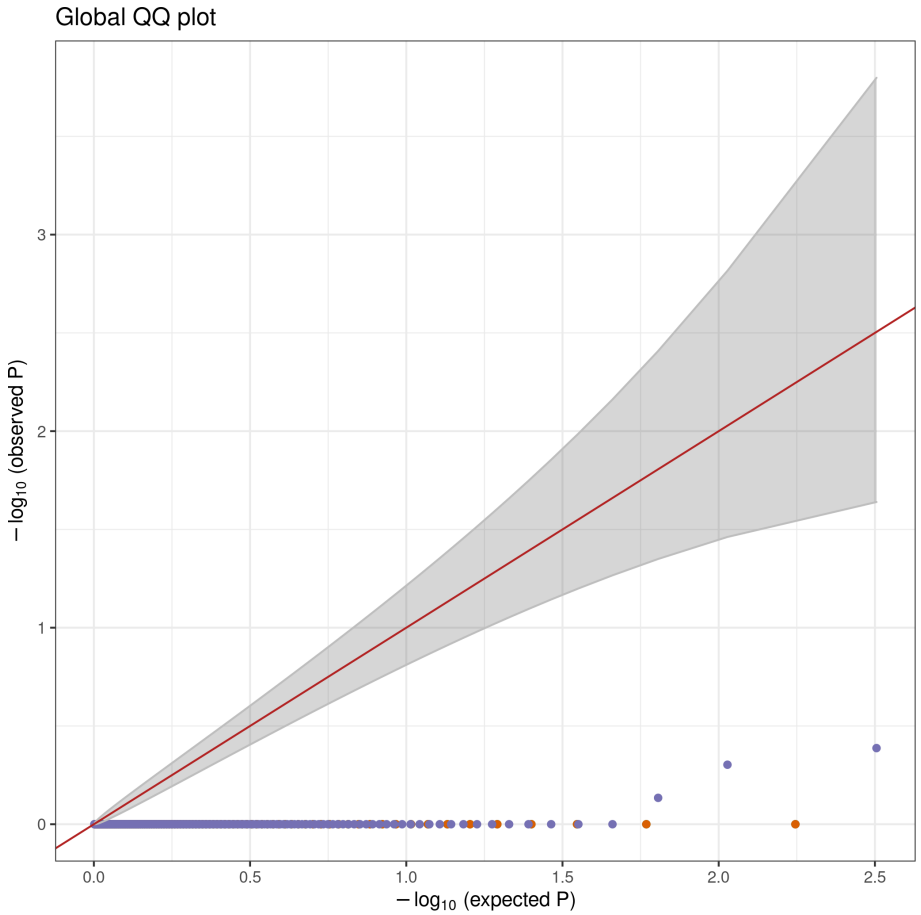
```
plotAberrantPerSample(fds)
```




```
# qq-plot for single junction
plotQQ(fds, result=res[1])
```




```
# global qq-plot (on gene level since aggregate=TRUE)
plotQQ(fds, aggregate=TRUE, global=TRUE)
```



References

[1] D. D. Pervouchine, D. G. Knowles, and R. Guigo. Intron-centric estimation of alternative splicing from RNA-seq data. *Bioinformatics*, 29(2):273–274, November 2012. URL: https://doi.org/10.1093/bioinformatics/bts678, doi:10.1093/bioinformatics/bts678.

[2] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, January 2013. URL: https://doi.org/10.1093/bioinformatics/bts635, doi:10.1093/bioinformatics/bts635.

[3] Santiago Marco-Sola, Michael Sammeth, Roderic Guigó, and Paolo Ribeca. The GEM mapper: fast, accurate and versatile alignment by filtration. *Nature Methods*, 9(12):1185–1188, October 2012. URL: https://doi.org/10.1038/nmeth.2221, doi:10.1038/nmeth.2221.

[4] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics*, 29(4):1165–1188, 2001. URL: https://projecteuclid.org/euclid.aos/1013699998, arXiv:0801.1095, doi:10.1214/aos/1013699998.

5 Session Info

Here is the output of `sessionInfo()` on the system on which this document was compiled:

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /home/biocbuild/bbs-3.14-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.14-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB              LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] org.Hs.eg.db_3.14.0
##  [2] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [3] GenomicFeatures_1.46.0
##  [4] AnnotationDbi_1.56.0
##  [5] FRASER_1.6.0
##  [6] SummarizedExperiment_1.24.0
##  [7] Biobase_2.54.0
##  [8] MatrixGenerics_1.6.0
##  [9] matrixStats_0.61.0
## [10] Rsamtools_2.10.0
## [11] Biostrings_2.62.0
## [12] XVector_0.34.0
## [13] GenomicRanges_1.46.0
## [14] GenomeInfoDb_1.30.0
```

```
## [15] IRanges_2.28.0
## [16] S4Vectors_0.32.0
## [17] BiocGenerics_0.40.0
## [18] data.table_1.14.2
## [19] knitr_1.36
## [20] BiocParallel_1.28.0
##
## loaded via a namespace (and not attached):
##   [1] backports_1.2.1           VGAM_1.1-5
##   [3] BiocFileCache_2.2.0       plyr_1.8.6
##   [5] lazyeval_0.2.2            splines_4.1.1
##   [7] ggplot2_3.3.5             digest_0.6.28
##   [9] foreach_1.5.1             htmltools_0.5.2
##  [11] magick_2.7.3              viridis_0.6.2
##  [13] fansi_0.5.0               magrittr_2.0.1
##  [15] checkmate_2.0.0           memoise_2.0.0
##  [17] BBmisc_1.11               BSgenome_1.62.0
##  [19] annotate_1.72.0           R.utils_2.11.0
##  [21] prettyunits_1.1.1         colorspace_2.0-2
##  [23] ggrepel_0.9.1             blob_1.2.2
##  [25] rappdirs_0.3.3            xfun_0.27
##  [27] dplyr_1.0.7               crayon_1.4.1
##  [29] RCurl_1.98-1.5            jsonlite_1.7.2
##  [31] genefilter_1.76.0         survival_3.2-13
##  [33] iterators_1.0.13          glue_1.4.2
##  [35] registry_0.5-1            gtable_0.3.0
##  [37] zlibbioc_1.40.0           webshot_0.5.2
##  [39] Rsubread_2.8.0            DelayedArray_0.20.0
##  [41] Rhdf5lib_1.16.0           HDF5Array_1.22.0
##  [43] scales_1.1.1              pheatmap_1.0.12
##  [45] DBI_1.1.1                 Rcpp_1.0.7
##  [47] viridisLite_0.4.0         xtable_1.8-4
##  [49] progress_1.2.2            bit_4.0.4
##  [51] htmlwidgets_1.5.4         httr_1.4.2
##  [53] RColorBrewer_1.1-2        ellipsis_0.3.2
##  [55] farver_2.1.0              pkgconfig_2.0.3
##  [57] XML_3.99-0.8              R.methodsS3_1.8.1
##  [59] dbplyr_2.1.1              locfit_1.5-9.4
##  [61] utf8_1.2.2                labeling_0.4.2
##  [63] tidyselect_1.1.1          rlang_0.4.12
##  [65] reshape2_1.4.4            PRROC_1.3.1
##  [67] munsell_0.5.0             tools_4.1.1
##  [69] cachem_1.0.6              cli_3.0.1
##  [71] generics_0.1.1            RSQLite_2.2.8
##  [73] evaluate_0.14             stringr_1.4.0
```

```
##  [75] fastmap_1.1.0             heatmaply_1.3.0
##  [77] yaml_2.2.1                bit64_4.0.5
##  [79] purrr_0.3.4               KEGGREST_1.34.0
##  [81] dendextend_1.15.1         nlme_3.1-153
##  [83] sparseMatrixStats_1.6.0   R.oo_1.24.0
##  [85] xml2_1.3.2                biomaRt_2.50.0
##  [87] BiocStyle_2.22.0          compiler_4.1.1
##  [89] plotly_4.10.0             filelock_1.0.2
##  [91] curl_4.3.2                png_0.1-7
##  [93] tibble_3.1.5              geneplotter_1.72.0
##  [95] stringi_1.7.5             highr_0.9
##  [97] lattice_0.20-45           Matrix_1.3-4
##  [99] vctrs_0.3.8               pillar_1.6.4
## [101] lifecycle_1.0.1           rhdf5filters_1.6.0
## [103] BiocManager_1.30.16       cowplot_1.1.1
## [105] bitops_1.0-7              seriation_1.3.1
## [107] rtracklayer_1.54.0        extraDistr_1.9.1
## [109] R6_2.5.1                  BiocIO_1.4.0
## [111] pcaMethods_1.86.0         TSP_1.1-11
## [113] gridExtra_2.3             codetools_0.2-18
## [115] assertthat_0.2.1          rhdf5_2.38.0
## [117] DESeq2_1.34.0             rjson_0.2.20
## [119] GenomicAlignments_1.30.0  GenomeInfoDbData_1.2.7
## [121] OUTRIDER_1.12.0           mgcv_1.8-38
## [123] parallel_4.1.1            hms_1.1.1
## [125] grid_4.1.1                tidyr_1.1.4
## [127] rmarkdown_2.11            DelayedMatrixStats_1.16.0
## [129] restfulr_0.0.13
```