FRASER: Find RAre Splicing Events in RNA-seq Data

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

[1] Technical University of Munich, Department of Informatics, Garching, Germany

November 4, 2026

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* version >= 1.99.0 in published research, please cite:

Scheller I, Lutz K, Mertes C, *et al.* **Improved detection of aberrant splicing with FRASER 2.0 and the Intron Jaccard Index**, Am J Hum Genet, 2023, *https://doi.org/10.1016/j.ajhg.2023.10.014*

For previous versions of *FRASER*, please cite:

Mertes C, Scheller I, Yepez V, *et al.* **Detection of aberrant splicing events in RNA-seq data using FRASER**, Nat Commun, 2021, *https://doi.org/10.1038/s41467-020-20573-7*

Package

FRASER 2.9.2

Contents

1 Introduction . . . 3

2 Quick guide to *FRASER* . . . 5

3 A detailed *FRASER* analysis . . . 11

3.1 Data preparation . . . 12
3.1.1 Creating a *FraserDataSet* and Counting reads . . . 12
3.1.2 Creating a *FraserDataSet* from existing count matrices . . . 15

3.2 Data preprocessing and QC . . . 18
3.2.1 Filtering . . . 18
3.2.2 Sample co-variation . . . 21

3.3 Detection of aberrant splicing events . . . 22
3.3.1 Fitting the splicing model . . . 22
3.3.2 Calling splicing outliers . . . 23
3.3.3 Interpreting the results table . . . 24

3.4 Finding splicing candidates in patients . . . 28

3.5 Saving and loading a *FraserDataSet* . . . 31

4 More details on *FRASER* . . . 32

4.1 Correction for confounders . . . 32
4.1.1 Finding the dimension of the latent space . . . 33

4.2 P-value calculation . . . 34

4.3 Result visualization . . . 36

References . . . 44

5 Session Info . . . 45

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.

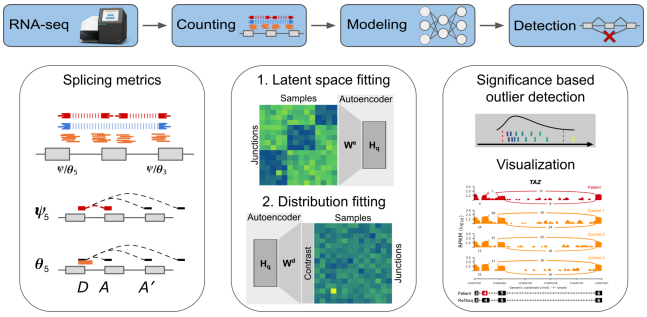


Figure 1: The *FRASER* splicing outlier detection workflow. The workflow starts with RNA-seq aligned reads and performs splicing outlier detection in three steps. First (left column), a splice site map is generated in an annotation-free fashion based on RNA-seq split reads. Split reads supporting exon-exon 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:

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

and

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

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.

From *FRASER* 2.0 on, only a single metric - the Intron Jaccard Index (Figure 2) - is used by default. The Intron Jaccard Index is more robust and allows to focus more on functionally relevant aberrant splicing events. It allows to detect all types of aberrant splicing previously detected using the three metrics (ψ_5, ψ_3, θ) within a single metric.

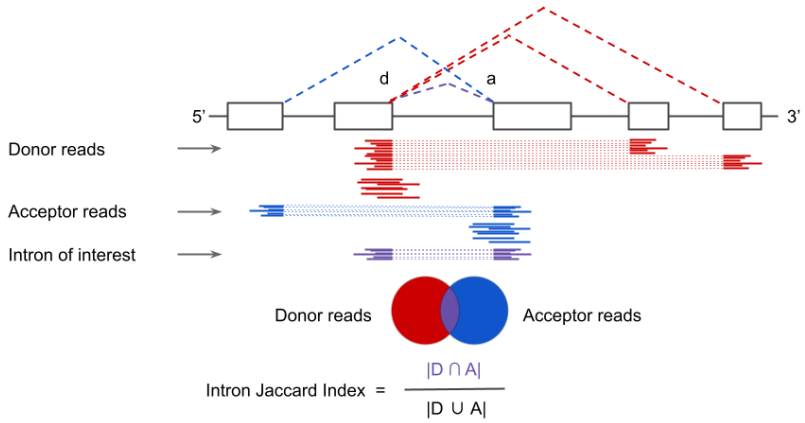


Figure 2: Overview over the Intron Jaccard Index, the splice metric used in *FRASER*2. The Intron Jaccard Index considers both split and nonsplit reads within a single metric and allows to detect all different types of aberrant splicing previously captured with either of the metrics ψ_5, ψ_3, θ.

The Intron Jaccard Index considers both split and nonsplit reads and is defined as the Jaccard index of the set of donor reads (reads sharing a donor site with the intron of interest and nonsplit reads at that donor site) and acceptor reads (reads sharing an acceptor site with the intron of interest and nonsplit reads at that acceptor site):

$$J(D,A) = \frac{n(D,A)}{\sum_{A'} n(D,A') + \sum_{D'} n(D',A) + n(D) + n(A) - n(D,A)} \quad (5)$$

2 Quick guide to *FRASER*

Here we show how to do an analysis with *FRASER*, starting from a sample annotation table and raw data (RNA-seq BAM files). First, we create a *FraserDataSet* object 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 lowly expressed. 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 4.3.

```
# load FRASER library
library(FRASER)

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

```
##
##         ==========     _____ _    _ ____  _____  ______          _____
##         =====         / ____| |  | |  _ \|  __ \|  ____|   /\   |  __ \
##           =====      | (___ | |  | | |_) | |__) | |__     /  \  | |  | |
##             ====      \___ \| |  | |  _ <|  _  /|  __|   / /\ \ | |  | |
##               ====    ____) | |__| | |_) | | \ \| |____ / ____ \| |__| |
##         ==========   |_____/ \____/|____/|_|  \_\______/_/    \_\_____/
##        Rsubread 2.27.7
##
## //========================== featureCounts setting ===========================\\
## ||                                                                            ||
## ||             Input files : 1 BAM file                                       ||
## ||                                                                            ||
## ||                           sample1.bam                                      ||
## ||                                                                            ||
## ||              Paired-end : yes                                              ||
## ||        Count read pairs : yes                                              ||
## ||              Annotation : R data.frame                                     ||
## ||      Dir for temp files : /tmp/Rtmp9RdJJB/Rbuild232a1a56b0b7/FRASER/vi ... ||
## ||                 Threads : 1                                                ||
## ||                   Level : meta-feature level                               ||
## ||      Multimapping reads : counted                                          ||
## || Multi-overlapping reads : counted                                          ||
## ||   Min overlapping bases : 10                                             ||
```

```
        ## ||                                                                            ||
        ## \\============================================================================//
        ##
        ## //================================= Running ==================================\\
        ## ||                                                                            ||
        ## || Load annotation file .Rsubread_UserProvidedAnnotation_pid9221 ...          ||
        ## ||    Features : 37                                                           ||
        ## ||    Meta-features : 37                                                      ||
        ## ||    Chromosomes/contigs : 2                                                 ||
        ## ||                                                                            ||
        ## || Process BAM file sample1.bam...                                           ||
        ## ||    Paired-end reads are included.                                          ||
        ## ||    Total alignments : 474                                                  ||
        ## ||    Successfully assigned alignments : 23 (4.9%)                            ||
        ## ||    Running time : 0.00 minutes                                             ||
        ## ||                                                                            ||
        ## || Write the final count table.                                              ||
        ## || Write the read assignment summary.                                        ||
        ## ||                                                                            ||
        ## \\============================================================================//
        ##
        ##          ==========     _____ _    _ ____  _____  ______          _____
        ##          =====         / ____| |  | |  _ \|  __ \|  ____|   /\   |  __ \
        ##            =====      | (___ | |  | | |_) | |__) | |__     /  \  | |  | |
        ##              ====      \___ \| |  | |  _ <|  _  /|  __|   / /\ \ | |  | |
        ##                ====    ____) | |__| | |_) | | \ \| |____ / ____ \| |__| |
        ##          ==========   |_____/ \____/|____/|_|  \_\______/_/    \_\_____/
        ##         Rsubread 2.27.7
        ##
        ## //========================== featureCounts setting ===========================\\
        ## ||                                                                            ||
        ## ||             Input files : 1 BAM file                                       ||
        ## ||                                                                            ||
        ## ||                           sample2.bam                                      ||
        ## ||                                                                            ||
        ## ||              Paired-end : yes                                              ||
        ## ||        Count read pairs : yes                                              ||
        ## ||              Annotation : R data.frame                                     ||
        ## ||      Dir for temp files : /tmp/Rtmp9RdJJB/Rbuild232a1a56b0b7/FRASER/vi ... ||
        ## ||                 Threads : 1                                                ||
        ## ||                   Level : meta-feature level                               ||
        ## ||      Multimapping reads : counted                                          ||
        ## || Multi-overlapping reads : counted                                          ||
        ## ||   Min overlapping bases : 10                                              ||
        ## ||                                                                            ||
```

```
## \\========================================================================//
##
## //================================ Running ================================\\
## ||                                                                        ||
## || Load annotation file .Rsubread_UserProvidedAnnotation_pid9221 ...      ||
## ||    Features : 37                                                       ||
## ||    Meta-features : 37                                                  ||
## ||    Chromosomes/contigs : 2                                             ||
## ||                                                                        ||
## || Process BAM file sample2.bam...                                        ||
## ||    Paired-end reads are included.                                      ||
## ||    Total alignments : 2455                                             ||
## ||    Successfully assigned alignments : 39 (1.6%)                        ||
## ||    Running time : 0.00 minutes                                         ||
## ||                                                                        ||
## || Write the final count table.                                           ||
## || Write the read assignment summary.                                     ||
## ||                                                                        ||
## \\========================================================================//
##
##          ==========     _____ _    _ ____  _____  ______          _____
##          =====         / ____| |  | |  _ \|  __ \|  ____|   /\   |  __ \
##            =====      | (___ | |  | | |_) | |__) | |__     /  \  | |  | |
##              ====      \___ \| |  | |  _ <|  _  /|  __|   / /\ \ | |  | |
##                ====    ____) | |__| | |_) | | \ \| |____ / ____ \| |__| |
##          ==========   |_____/ \____/|____/|_|  \_\______/_/    \_\_____/
##         Rsubread 2.27.7
##
## //========================== featureCounts setting ===========================\\
## ||                                                                        ||
## ||             Input files : 1 BAM file                                   ||
## ||                                                                        ||
## ||                           sample3.bam                                  ||
## ||                                                                        ||
## ||              Paired-end : yes                                          ||
## ||        Count read pairs : yes                                          ||
## ||              Annotation : R data.frame                                 ||
## ||      Dir for temp files : /tmp/Rtmp9RdJJB/Rbuild232a1a56b0b7/FRASER/vi ... ||
## ||                 Threads : 1                                            ||
## ||                   Level : meta-feature level                           ||
## ||      Multimapping reads : counted                                      ||
## || Multi-overlapping reads : counted                                      ||
## ||   Min overlapping bases : 10                                          ||
## ||                                                                        ||
## \\========================================================================//
```

```
##
## //================================= Running ==================================\\
## ||                                                                            ||
## || Load annotation file .Rsubread_UserProvidedAnnotation_pid9221 ...          ||
## ||    Features : 37                                                           ||
## ||    Meta-features : 37                                                      ||
## ||    Chromosomes/contigs : 2                                                 ||
## ||                                                                            ||
## || Process BAM file sample3.bam...                                            ||
## ||    Paired-end reads are included.                                          ||
## ||    Total alignments : 1918                                                 ||
## ||    Successfully assigned alignments : 37 (1.9%)                            ||
## ||    Running time : 0.00 minutes                                             ||
## ||                                                                            ||
## || Write the final count table.                                               ||
## || Write the read assignment summary.                                         ||
## ||                                                                            ||
## \\============================================================================//
```

```
fds
```

```
## ------------------- Sample data table -----------------
## # A tibble: 3 x 7
##   sampleID bamFile             condition gene  pairedEnd strand SeqLevelStyle
##   <chr>    <chr>                   <int> <chr> <lgl>      <int> <chr>
## 1 sample1  /tmp/Rtmp9RdJJB/Ri~         1 TIMM~ TRUE           0 UCSC
## 2 sample2  /tmp/Rtmp9RdJJB/Ri~         3 CLPP  TRUE           0 UCSC
## 3 sample3  /tmp/Rtmp9RdJJB/Ri~         2 MCOL~ TRUE           0 UCSC
##
## Number of samples:      3
## Number of junctions:    48
## Number of splice sites: 37
## assays(2): rawCountsJ rawCountsSS
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
```

```
## bamWhich: 0 ranges
## bamWhat:
## bamMapqFilter: 0

# compute stats
fds <- calculatePSIValues(fds)

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

# we provide two ways to annotate 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)

# fit the splicing model for each metric
# with a specific latentspace dimension
fds <- FRASER(fds, q=c(jaccard=2))

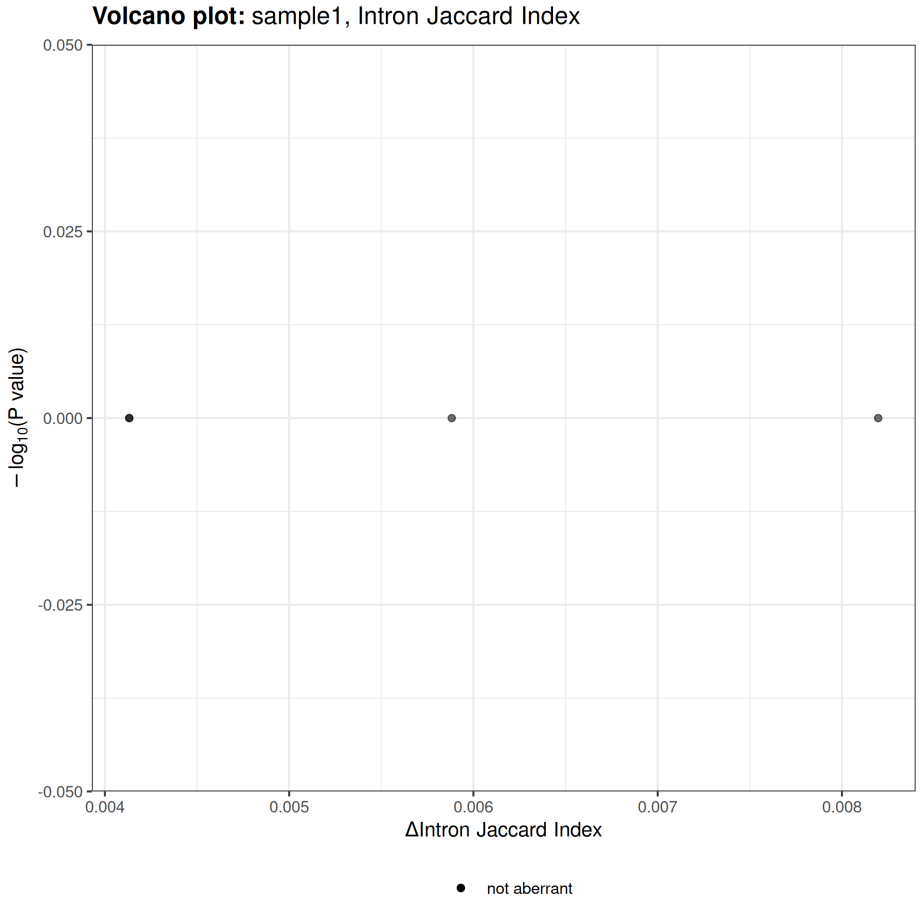
# Alternatively, we also provide a way to use BioMart for the annotation:
# fds <- annotateRanges(fds)

# get results: we recommend to use an FDR cutoff of 0.05, but due to the small
# dataset size, we extract all events and their associated values
# eg: res <- results(fds, padjCutoff=0.05, deltaPsiCutoff=0.1)
res <- results(fds, all=TRUE)
res

## GRanges object with 57 ranges and 15 metadata columns:
##        seqnames              ranges strand |                names
##           <Rle>           <IRanges>  <Rle> |          <character>
##    [1]    chr19     7590053-7591324      * | Intron Jaccard Index
##    [2]    chr19     7590053-7591324      * | Intron Jaccard Index
##    [3]    chr19     7590053-7591324      * | Intron Jaccard Index
##    [4]    chr19     7591493-7591646      * | Intron Jaccard Index
##    [5]    chr19     7591493-7591646      * | Intron Jaccard Index
##    ...      ...                 ...    ... .                  ...
##   [53]     chr3 119234787-119236051      * | Intron Jaccard Index
##   [54]     chr3 119234787-119236051      * | Intron Jaccard Index
##   [55]     chr3 119236163-119242452      * | Intron Jaccard Index
##   [56]     chr3 119236163-119242452      * | Intron Jaccard Index
##   [57]     chr3 119236163-119242452      * | Intron Jaccard Index
```

```
##           sampleID   hgncSymbol        type    pValue   padjust  psiValue
##         <character>  <character> <character> <numeric> <numeric> <numeric>
##    [1]     sample1       MCOLN1     jaccard         1         1      0.82
##    [2]     sample2       MCOLN1     jaccard         1         1      0.95
##    [3]     sample3       MCOLN1     jaccard         1         1      0.83
##    [4]     sample1       MCOLN1     jaccard         1         1      1.00
##    [5]     sample2       MCOLN1     jaccard         1         1      1.00
##    ...         ...          ...         ...       ...       ...       ...
##   [53]     sample2      TIMMDC1     jaccard         1         1      0.01
##   [54]     sample3      TIMMDC1     jaccard         1         1      0.01
##   [55]     sample1 TIMMDC1;CD80     jaccard         1         1      1.00
##   [56]     sample2 TIMMDC1;CD80     jaccard         1         1      0.98
##   [57]     sample3 TIMMDC1;CD80     jaccard         1         1      1.00
##         deltaPsi    counts totalCounts meanCounts meanTotalCounts
##        <numeric> <integer>   <numeric>  <numeric>       <numeric>
##    [1]      0.01         9          11      11.33           12.67
##    [2]      0.00        20          21      11.33           12.67
##    [3]      0.01         5           6      11.33           12.67
##    [4]      0.01        12          12      12.00           12.00
##    [5]      0.00        21          21      12.00           12.00
##    ...       ...       ...         ...        ...             ...
##   [53]         0         3         296      12.67          219.33
##   [54]         0         4         329      12.67          219.33
##   [55]         0        24          24     181.33          183.00
##   [56]         0       256         260     181.33          183.00
##   [57]         0       264         265     181.33          183.00
##        nonsplitCounts nonsplitProportion nonsplitProportion_99quantile
##             <numeric>          <numeric>                      <numeric>
##    [1]              2               0.18                           0.18
##    [2]              0               0.00                           0.18
##    [3]              1               0.17                           0.18
##    [4]              0               0.00                           0.00
##    [5]              0               0.00                           0.00
##    ...            ...                ...                            ...
##   [53]              0               0.00                           0.00
##   [54]              0               0.00                           0.00
##   [55]              0               0.00                           0.01
##   [56]              3               0.01                           0.01
##   [57]              0               0.00                           0.01
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths

# result visualization, aggregate=TRUE means that results are aggregated at the gene level
plotVolcano(fds, sampleID="sample1", type="jaccard", 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. Calculating P-values 4.2
5. Visualizing the results 4.3

Steps 3 and 4 are wrapped up in one function `FRASER`, but each step can be called individually and parametrized. 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 an 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 a unique sample name and the path to the BAM file. Additionally groups can be specified for the P-value calculations. If a **NA** is assigned, no P-values will be calculated. An example sample table is given within the package:

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

```
##    sampleID                bamFile group    gene pairedEnd
##      <char>                 <char> <int>  <char>    <lgcl>
## 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="FRASER_output")

# 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/Rtmp9RdJJB/Rinst232a3d38615/FRASER/ext~     1 TIMM~ TRUE
## 2 sample2  /tmp/Rtmp9RdJJB/Rinst232a3d38615/FRASER/ext~     3 CLPP  TRUE
## 3 sample3  /tmp/Rtmp9RdJJB/Rinst232a3d38615/FRASER/ext~     2 MCOL~ TRUE
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat:
## 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 6
##   sampleID bamFile                           condition gene  pairedEnd strand
##   <chr>    <chr>                                 <int> <chr> <lgl>      <int>
## 1 sample1  /tmp/Rtmp9RdJJB/Rinst232a3d38615~         1 TIMM~ TRUE           0
## 2 sample2  /tmp/Rtmp9RdJJB/Rinst232a3d38615~         3 CLPP  TRUE           0
```

```
## 3 sample3  /tmp/Rtmp9RdJJB/Rinst232a3d38615~         2 MCOL~ TRUE           0
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat:
## bamMapqFilter: 0
```

Counting the reads is straightforward 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 are counted. The resulting *FraserDataSet* object contains two *SummarizedExperiment* objects, one for the junctions and one for 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 <- countRNADData(settings)
fds
```

```
## ------------------- Sample data table ----------------
## # A tibble: 3 x 6
##   sampleID bamFile                          condition gene  pairedEnd strand
##   <chr>    <chr>                                <int> <chr> <lgl>      <int>
## 1 sample1  /tmp/Rtmp9RdJJB/Rinst232a3d38615~         1 TIMM~ TRUE           0
## 2 sample2  /tmp/Rtmp9RdJJB/Rinst232a3d38615~         3 CLPP  TRUE           0
## 3 sample3  /tmp/Rtmp9RdJJB/Rinst232a3d38615~         2 MCOL~ TRUE           0
```

```
##
## Number of samples:      3
## Number of junctions:    48
## Number of splice sites: 37
## assays(2): rawCountsJ rawCountsSS
##
## ---------------------- Settings -----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat:
## 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 annotation for precalculated count matrices
sampleTable <- fread(system.file("extdata", "sampleTable_countTable.tsv",
        package="FRASER", mustWork=TRUE))
head(sampleTable)
```

```
##    sampleID                bamFile group    gene
##      <char>                 <char> <int>  <char>
## 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
##      <char>   <int>   <int>  <int> <char>   <int>   <int>   <int>   <int>
## 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
##      <int>   <int>   <int>   <int>   <int>    <int>    <int>    <int>
## 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
##      <int> <int>
## 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
##      <char>   <int>   <int> <int> <char>        <int> <char>   <int>   <int>
## 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
##      <int>   <int>   <int>   <int>   <int>   <int>   <int>    <int>    <int>
## 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
```

```
##       <int>
## 1:        0
## 2:        0
## 3:        0
## 4:        0
## 5:        0
## 6:        0
```

```
# create FRASER object
fds <- FraserDataSet(colData=sampleTable, junctions=junctionCts,
        spliceSites=spliceSiteCts, workingDir="FRASER_output")
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:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
```

```
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat:
## 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].

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 obtain better results, at least 50 samples should be sequenced and they should be processed with the same protocol and originated from the same tissue.

3.2.1 Filtering

Before filtering the data, we have to compute the main splicing metrics: the ψ-value (Percent Spliced In) and the Intron Jaccard Index.

```
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(15): rawCountsJ psi5 ... rawOtherCounts_theta delta_theta
```

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

```
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat:
## bamMapqFilter: 0
```

Now we can filter down the number of junctions we want to test later on.

Currently, we suggest keeping only junctions which support the following:

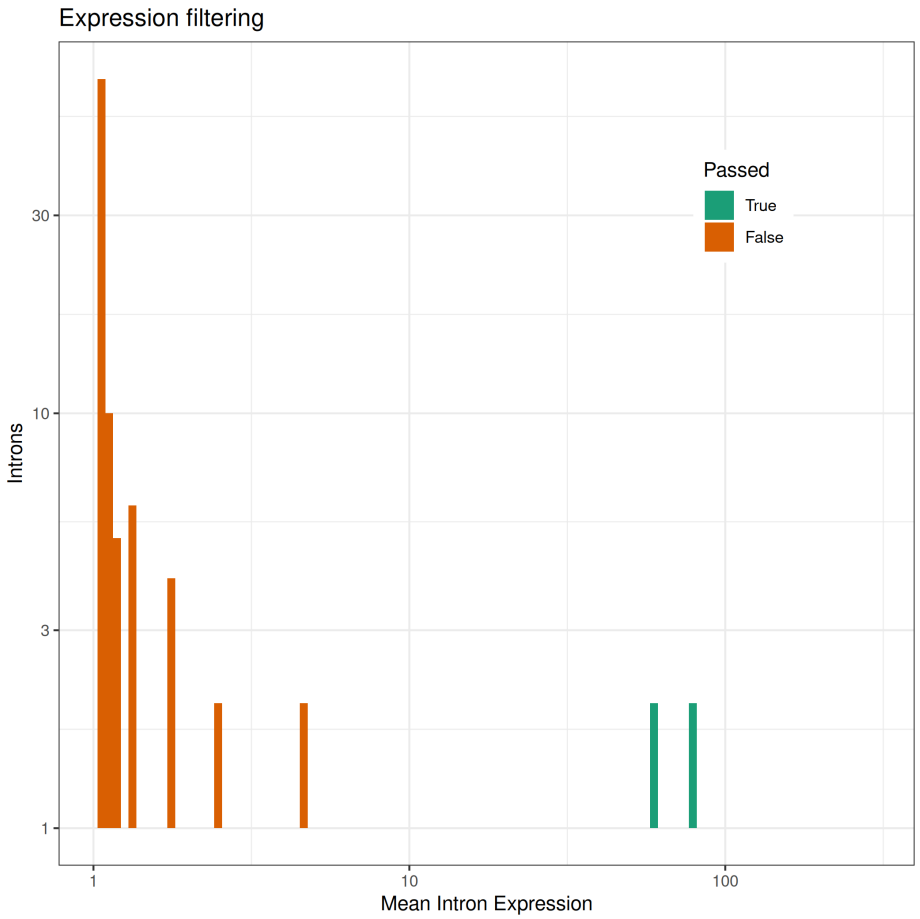
- At least one sample has 20 (or more) reads
- 25% (or more) of the samples have at least 10 reads

Furthemore one could filter for:

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

```
fds <- filterExpressionAndVariability(fds, minDeltaPsi=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(15): rawCountsJ psi5 ... rawOtherCounts_theta delta_theta
##
## ---------------------- Settings ----------------------
## Analysis name:               Data Analysis
## Analysis is strand specific: no
## Working directory:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat:
## bamMapqFilter: 0
```

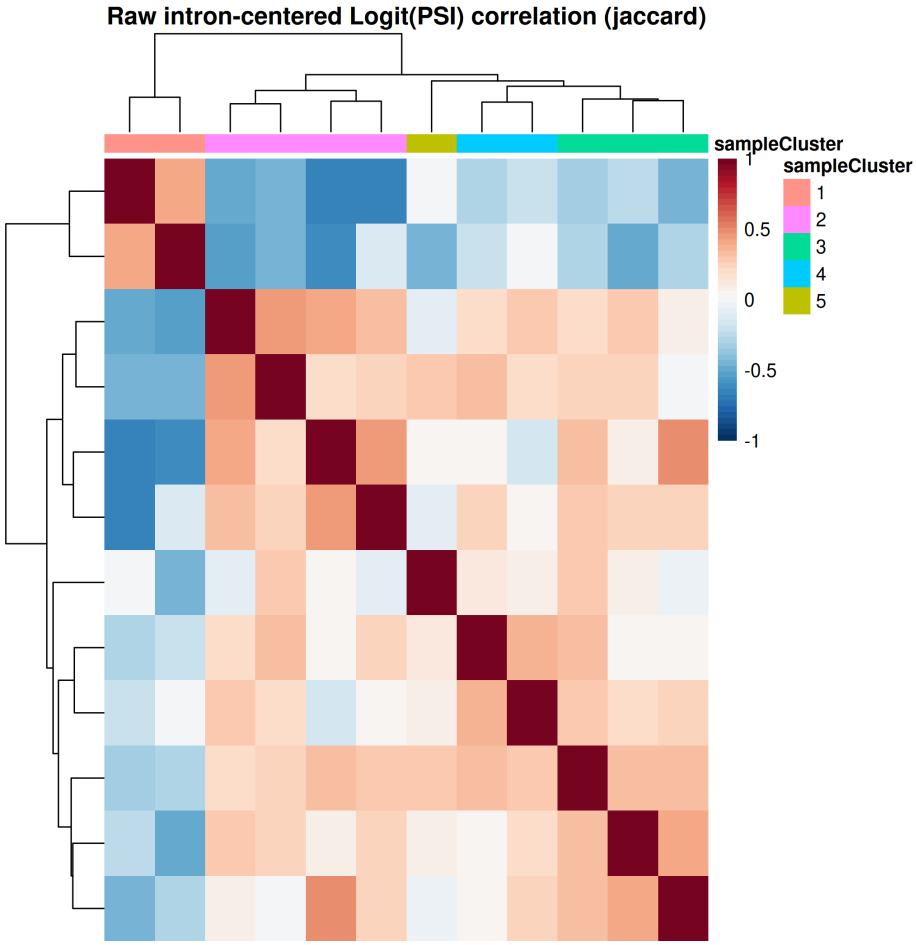
```
# filtered_fds not further used for this tutorial because the example dataset
# is otherwise too small
```

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.

However, we do see strong sample co-variation across different tissues and cohorts. Let's have a look into our demo data to see if we it has 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="jaccard", 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="jaccard", 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 with our analysis. Let's start with the first step in the aberrant splicing detection: the model fitting.

3.3.1 Fitting the splicing model

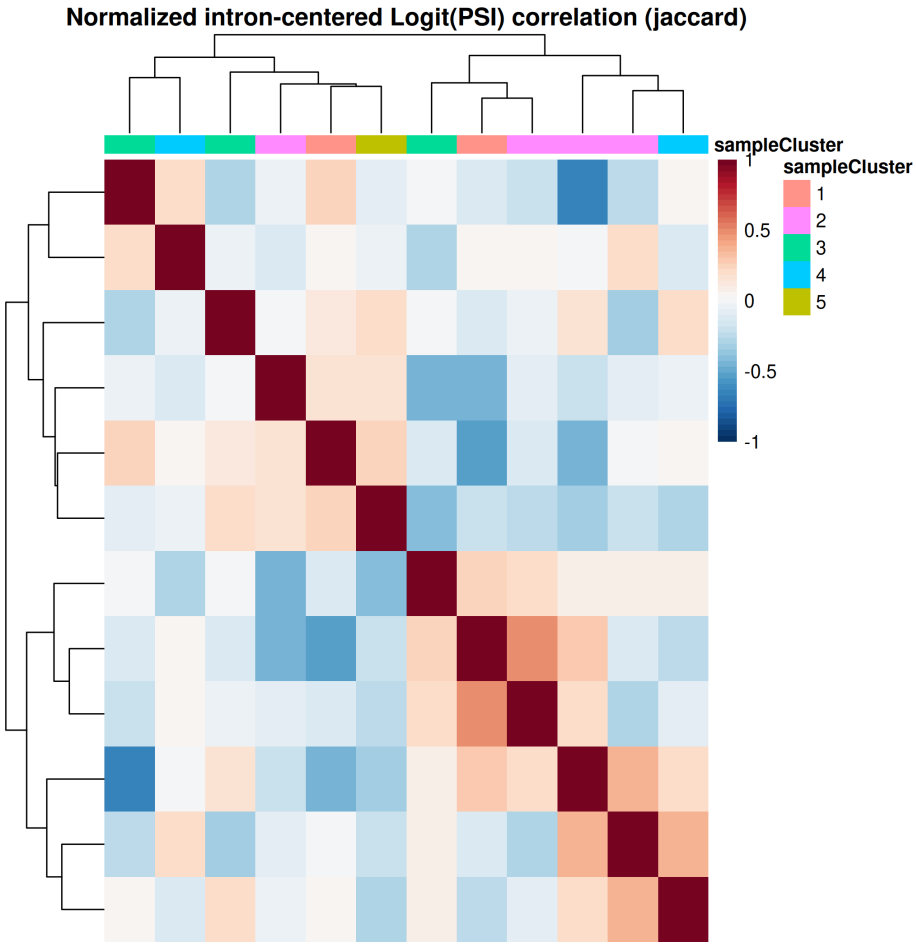
During the fitting procedure, we will normalize the data and correct for confounding effects by using a denoising autoencoder. Here we use a predefined latent space with a dimension $q = 10$. Using the correct dimension is crucial to have the best

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 for all ψ types. For more details see section 4.

```
# This is computational heavy on real datasets and can take some hours
fds <- FRASER(fds, q=c(jaccard=3))
```

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="jaccard", normalized=TRUE, logit=TRUE)
```



3.3.2 Calling splicing outliers

Before we extract the results, we should add HGNC symbols to the junctions. *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 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 results 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 jaccard for Intron Jaccard Index or psi5 for ψ_5, psi3 for ψ_3 or theta for θ)
- pValue, padjust: the p-value and adjusted p-value (FDR) of this event (at intron or splice site level depending on metric)
- pValueGene, padjustGene: only present in the gene-level results table, gives the p-value and FDR adjusted p-value at gene-level
- psiValue: the value of the splice metric (see 'type' column for the name of the metric) 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
- counts, totalCounts: the count (k) and total count (n) of the splice junction or site for the sample where it is detected as aberrant
- 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
- nonsplitCounts, nonsplitProportion: only present for the Intron Jaccard Index. States the sum of nonsplit counts overlapping either the donor or acceptor site of the outlier intron for the sample where it is detected as aberrant; and their proportion out of the total counts (N). A high nonsplitProportion indicates possible (partial) intron retention.
- FDR_set The set of genes on which FDR correction is applied. If not otherwise specified, FDR correction is transcriptome-wide.

Please refer to section 1 for more information about the Intron Jaccard Index metric (or the previous 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. As the Intron Jaccard Index combines the three metrics, an aberrant Intron Jaccard value can indicate any of the above described cases. We recommend inspecting the outliers using IGV. *FRASER*2 also provides the function `plotBamCoverageFromResultTable` to create a sashimi plot for an outlier in the results table directly in R (if paths to bam files are available in the *FraserDataSet* object).

```
# for visualization purposes for this tutorial, no cutoffs were used
res <- results(fds, all=TRUE)
res
```

```
## GRanges object with 1476 ranges and 15 metadata columns:
##          seqnames              ranges strand |                  names
##             <Rle>           <IRanges>  <Rle> |            <character>
##      [1]    chr19     7592515-7592749      * | Intron Jaccard Index..
##      [2]     chr3 119217436-119219541      * | Intron Jaccard Index..
##      [3]     chr3 119236163-119242452      * | Intron Jaccard Index..
##      [4]     chr3 119217567-119217621      * | Intron Jaccard Index..
##      [5]     chr3 119222869-119236051      * | Intron Jaccard Index..
##      ...      ...                  ...    ... .                    ...
##   [1472]    chr19     7703987-7704616      * | Intron Jaccard Index..
##   [1473]    chr19     7703987-7704616      * | Intron Jaccard Index..
##   [1474]    chr19     7703987-7704616      * | Intron Jaccard Index..
##   [1475]    chr19     7703987-7704616      * | Intron Jaccard Index..
##   [1476]    chr19     7703987-7704616      * | Intron Jaccard Index..
##             sampleID   hgncSymbol        type    pValue   padjust  psiValue
##          <character>  <character> <character> <numeric> <numeric> <numeric>
##      [1]     sample3       MCOLN1     jaccard  0.010087         1      0.14
##      [2]     sample9      TIMMDC1     jaccard  0.012551         1      0.04
##      [3]    sample11 TIMMDC1;CD80     jaccard  0.015124         1      0.98
##      [4]     sample7      TIMMDC1     jaccard  0.022096         1      0.01
```

```
##      [5]     sample7      TIMMDC1     jaccard  0.026414         1      0.01
##      ...         ...          ...         ...       ...       ...       ...
##   [1472]     sample5       STXBP2     jaccard         1         1       NaN
##   [1473]     sample6       STXBP2     jaccard         1         1         1
##   [1474]     sample7       STXBP2     jaccard         1         1       NaN
##   [1475]     sample8       STXBP2     jaccard         1         1       NaN
##   [1476]     sample9       STXBP2     jaccard         1         1       NaN
##           deltaPsi    counts totalCounts meanCounts meanTotalCounts
##          <numeric> <integer>   <numeric>  <numeric>       <numeric>
##      [1]     -0.45         3          22      92.50          103.08
##      [2]      0.03        12         303       4.83          364.92
##      [3]     -0.02       570         581     363.00          366.42
##      [4]      0.01         4         337       0.33          421.08
##      [5]      0.01         4         533       1.00          591.75
##      ...       ...       ...         ...        ...             ...
##   [1472]       NaN         0           0       0.08            0.08
##   [1473]      0.26         1           1       0.08            0.08
##   [1474]       NaN         0           0       0.08            0.08
##   [1475]       NaN         0           0       0.08            0.08
##   [1476]       NaN         0           0       0.08            0.08
##          nonsplitCounts nonsplitProportion nonsplitProportion_99quantile
##               <numeric>          <numeric>                     <numeric>
##      [1]             19               0.86                          0.79
##      [2]             64               0.21                          0.33
##      [3]              8               0.01                          0.02
##      [4]            333               0.99                          1.00
##      [5]              1               0.00                          0.01
##      ...            ...                ...                           ...
##   [1472]              0                NaN                            NA
##   [1473]              0                  0                            NA
##   [1474]              0                NaN                            NA
##   [1475]              0                NaN                            NA
##   [1476]              0                NaN                            NA
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

```
# for the gene level pvalues, gene symbols need to be added to the fds object
# before calling the calculatePadjValues function (part of FRASER() function)
# as we previously called FRASER() before annotating genes, we run it again here
fds <- calculatePadjValues(fds, type="jaccard", geneLevel=TRUE)
# generate gene-level results table (if gene symbols have been annotated)
res_gene <- results(fds, aggregate=TRUE, all=TRUE)
res_gene
```

```
## GRanges object with 540 ranges and 17 metadata columns:
##         seqnames              ranges strand |                 names
```

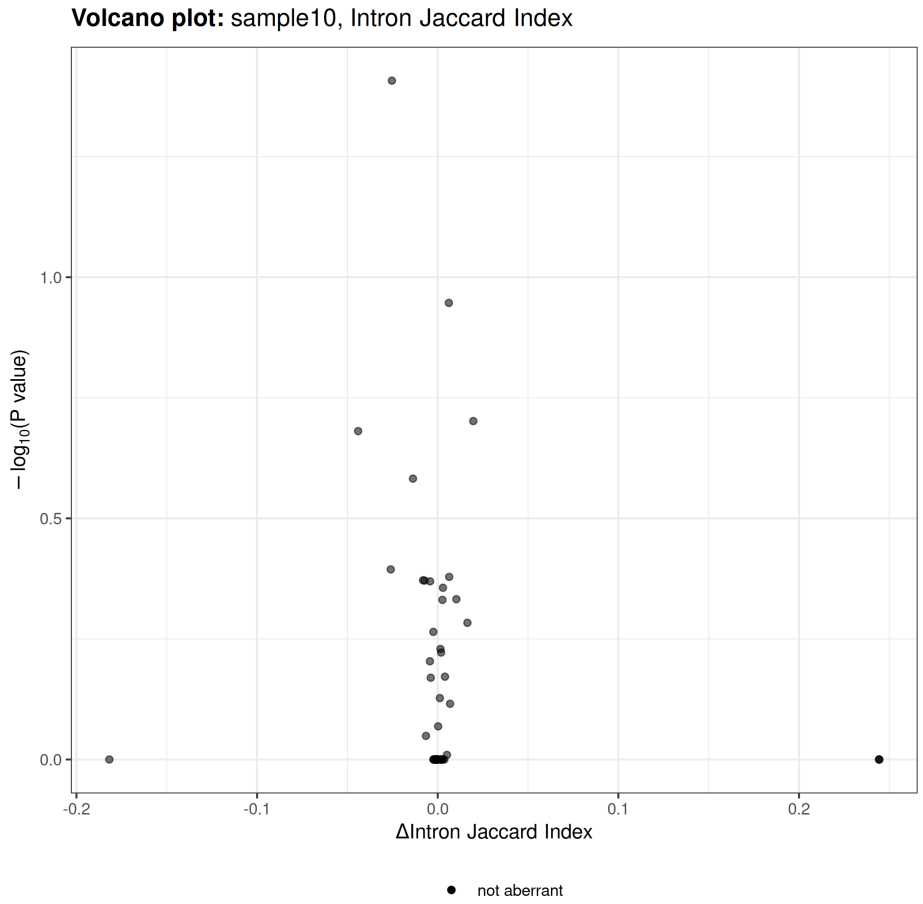
```
##           <Rle>           <IRanges>  <Rle> |           <character>
##     [1]    chr19     7592515-7592749      * | Intron Jaccard Index..
##     [2]     chr3 119217436-119219541      * | Intron Jaccard Index..
##     [3]     chr3 119236163-119242452      * | Intron Jaccard Index..
##     [4]     chr3 119236163-119242452      * | Intron Jaccard Index..
##     [5]     chr3 119217567-119217621      * | Intron Jaccard Index..
##     ...      ...                 ...    ... .                   ...
##   [536]    chr19     7585706-7855730      * | Intron Jaccard Index..
##   [537]    chr19     7585706-7855730      * | Intron Jaccard Index..
##   [538]    chr19     7585706-7855730      * | Intron Jaccard Index..
##   [539]    chr19     7598540-7598644      * | Intron Jaccard Index..
##   [540]    chr19     7598540-7598644      * | Intron Jaccard Index..
##            sampleID  hgncSymbol        type    pValue   padjust  psiValue
##         <character> <character> <character> <numeric> <numeric> <numeric>
##     [1]     sample3      MCOLN1     jaccard  0.010087         1      0.14
##     [2]     sample9     TIMMDC1     jaccard  0.012551         1      0.04
##     [3]    sample11        CD80     jaccard  0.015124         1      0.98
##     [4]    sample11     TIMMDC1     jaccard  0.015124         1      0.98
##     [5]     sample7     TIMMDC1     jaccard  0.022096         1      0.01
##     ...         ...         ...         ...       ...       ...       ...
##   [536]    sample12     TRAPPC5     jaccard         1         1         1
##   [537]    sample12        XAB2     jaccard         1         1         1
##   [538]    sample12      ZNF358     jaccard         1         1         1
##   [539]     sample5      PNPLA6     jaccard         1         1         1
##   [540]     sample7      PNPLA6     jaccard         1         1         1
##          deltaPsi    counts totalCounts meanCounts meanTotalCounts
##         <numeric> <integer>   <numeric>  <numeric>       <numeric>
##     [1]     -0.45         3          22      92.50          103.08
##     [2]      0.03        12         303       4.83          364.92
##     [3]     -0.02       570         581     363.00          366.42
##     [4]     -0.02       570         581     363.00          366.42
##     [5]      0.01         4         337       0.33          421.08
##     ...       ...       ...         ...        ...             ...
##   [536]      0.45         1           1       0.08            0.25
##   [537]      0.45         1           1       0.08            0.25
##   [538]      0.45         1           1       0.08            0.25
##   [539]      0.01         7           7      41.33           42.42
##   [540]      0.01        22          22      41.33           42.42
##         nonsplitCounts nonsplitProportion nonsplitProportion_99quantile
##              <numeric>          <numeric>                     <numeric>
##     [1]             19               0.86                          0.79
##     [2]             64               0.21                          0.33
##     [3]              8               0.01                          0.02
##     [4]              8               0.01                          0.02
##     [5]            333               0.99                          1.00
```

```
##     ...            ...               ...                          ...
##   [536]             0                 0                           NA
##   [537]             0                 0                           NA
##   [538]             0                 0                           NA
##   [539]             0                 0                         0.08
##   [540]             0                 0                         0.08
##         pValueGene padjustGene
##          <numeric>   <numeric>
##     [1]    0.55480           1
##     [2]    0.71538           1
##     [3]    0.21174           1
##     [4]    0.86207           1
##     [5]    1.00000           1
##     ...        ...         ...
##   [536]          1           1
##   [537]          1           1
##   [538]          1           1
##   [539]          1           1
##   [540]          1           1
##   -------
##   seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

3.4 Finding splicing candidates in patients

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

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



Which are the splicing events in detail?

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

```
## GRanges object with 123 ranges and 15 metadata columns:
##         seqnames              ranges strand |                 names
##            <Rle>           <IRanges>  <Rle> |           <character>
##     [1]     chr3 119217438-119219541      * | Intron Jaccard Index..
##     [2]     chr3 119234787-119236051      * | Intron Jaccard Index..
##     [3]    chr19     7594599-7595320      * | Intron Jaccard Index..
##     [4]    chr19     7593590-7593706      * | Intron Jaccard Index..
##     [5]    chr19     7593144-7593482      * | Intron Jaccard Index..
##     ...      ...                 ...    ... .                   ...
##   [119]    chr19     7615994-7616247      * | Intron Jaccard Index..
##   [120]    chr19     7616324-7618757      * | Intron Jaccard Index..
##   [121]    chr19     7625651-7625899      * | Intron Jaccard Index..
##   [122]    chr19     7690931-7691021      * | Intron Jaccard Index..
##   [123]    chr19     7703987-7704616      * | Intron Jaccard Index..
##            sampleID  hgncSymbol        type    pValue   padjust  psiValue
##         <character> <character> <character> <numeric> <numeric> <numeric>
```

```
##     [1]    sample10     TIMMDC1     jaccard  0.039124         1      0.02
##     [2]    sample10     TIMMDC1     jaccard  0.113040         1      0.01
##     [3]    sample10      MCOLN1     jaccard  0.198770         1      0.06
##     [4]    sample10      MCOLN1     jaccard  0.208480         1      0.92
##     [5]    sample10      MCOLN1     jaccard  0.261590         1      0.98
##     ...         ...         ...         ...       ...       ...       ...
##   [119]    sample10      PNPLA6     jaccard         1         1       NaN
##   [120]    sample10      PNPLA6     jaccard         1         1       NaN
##   [121]    sample10      PNPLA6     jaccard         1         1       NaN
##   [122]    sample10        XAB2     jaccard         1         1       NaN
##   [123]    sample10      STXBP2     jaccard         1         1       NaN
##          deltaPsi    counts totalCounts meanCounts meanTotalCounts
##         <numeric> <integer>   <numeric>  <numeric>       <numeric>
##     [1]     -0.03         9         478       3.67          367.67
##     [2]      0.01         4         420      10.50          306.17
##     [3]      0.02        14         250       4.92          146.75
##     [4]     -0.04       101         110      70.58           78.83
##     [5]     -0.01       145         148      93.58           97.25
##     ...       ...       ...         ...        ...             ...
##   [119]       NaN         0           0       0.08            0.08
##   [120]       NaN         0           0       0.08            0.08
##   [121]       NaN         0           0       0.25            0.25
##   [122]       NaN         0           0       0.08            0.08
##   [123]       NaN         0           0       0.08            0.08
##         nonsplitCounts nonsplitProportion nonsplitProportion_99quantile
##              <numeric>          <numeric>                       <numeric>
##     [1]           135               0.28                            0.33
##     [2]             0               0.00                            0.01
##     [3]           112               0.45                            0.52
##     [4]             8               0.07                            0.16
##     [5]             2               0.01                            0.17
##     ...           ...                ...                             ...
##   [119]             0                NaN                              NA
##   [120]             0                NaN                              NA
##   [121]             0                NaN                              NA
##   [122]             0                NaN                              NA
##   [123]             0                NaN                              NA
##   -------
##   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="jaccard", result=sampleRes[9]) # plots the 9th row
plotSpliceMetricRank(fds, type="jaccard", result=sampleRes[9])
plotExpectedVsObservedPsi(fds, result=sampleRes[9])
```

3.5 Saving and loading a *FraserDataSet*

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

```
# saving a fds
workingDir(fds) <- "FRASER_output"
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(19): rawCountsJ psi5 ... rawOtherCounts_theta delta_theta
##
## ---------------------- Settings ----------------------
## Analysis name:               ExampleAnalysis
## Analysis is strand specific: no
## Working directory:           'FRASER_output'
##
## ------------------- BAM parameters -------------------
## class: ScanBamParam
## bamFlag (NA unless specified):
## bamSimpleCigar: FALSE
## bamReverseComplement: FALSE
## bamTag:
## bamTagFilter:
## bamWhich: 0 ranges
## bamWhat:
## 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 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 iterations to speed the calculation up for the vignette
# the default is 15 iterations
fds <- fit(fds, q=3, type="jaccard", implementation="PCA-BB-Decoder",
            iterations=2)

##
## TRUE
##  123
## [1] "Finished with fitting the E matrix. Starting now with the D fit. ..."
## [1] "Wed Nov  4 20:32:16 2026: Iteration: final_1 loss: 0.781999986797768 (mean); 4.668106257069
## [1] "Wed Nov  4 20:32:16 2026: Iteration: final_2 loss: 0.797610284965922 (mean); 4.208203330032
## [1] "2 Final betabin-AE loss: 0.797610284965922"
```

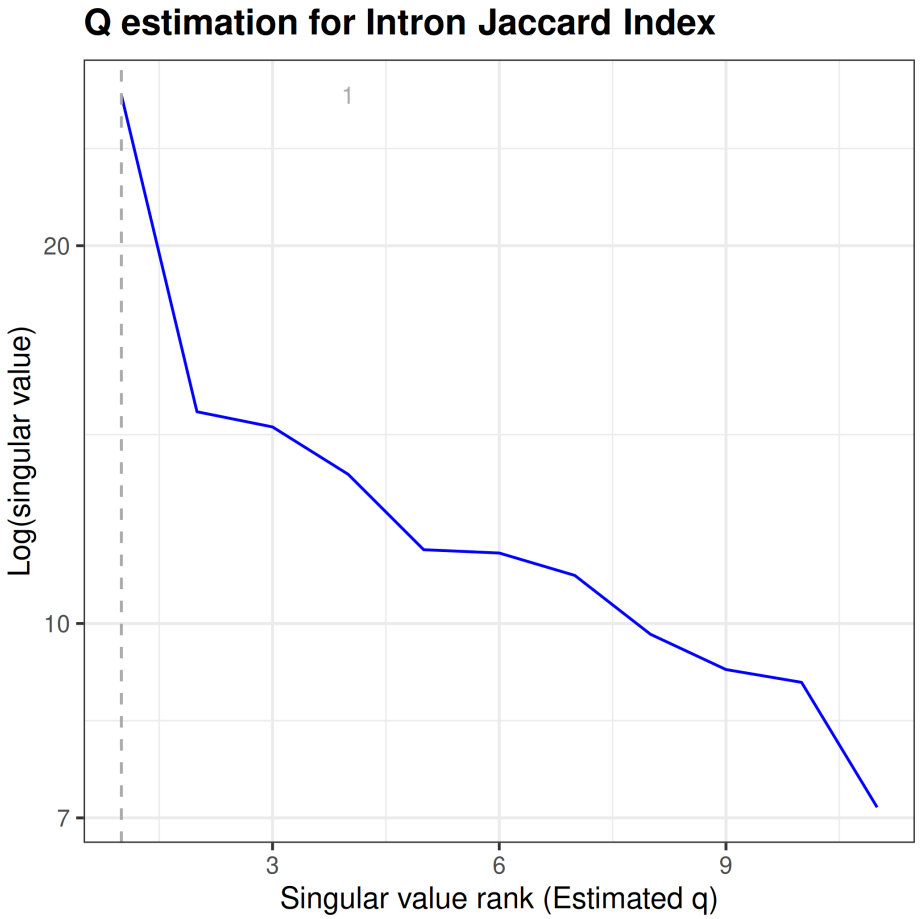
4.1.1 Finding the dimension of the latent space

For the previous call, the dimension q of the latent space has been fixed. Since working with the correct q is very important, the *FRASER* package also provides the function `estimateBestQ` that can be used to estimate the dimension q of the latent space of the data. Per default, it uses the deterministic Optimal Hard Thresholding (OHT) method established by Gavish and Donoho[4]. Alternatively, a hyperparameter optimization can be performed that 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 is quite CPU-intensive, we only show it here for a subset of the dataset and recommend to use the faster OHT approach. The results from the estimation of the optimal latent space dimension can be visualized with the function `plotEncDimSearch`.

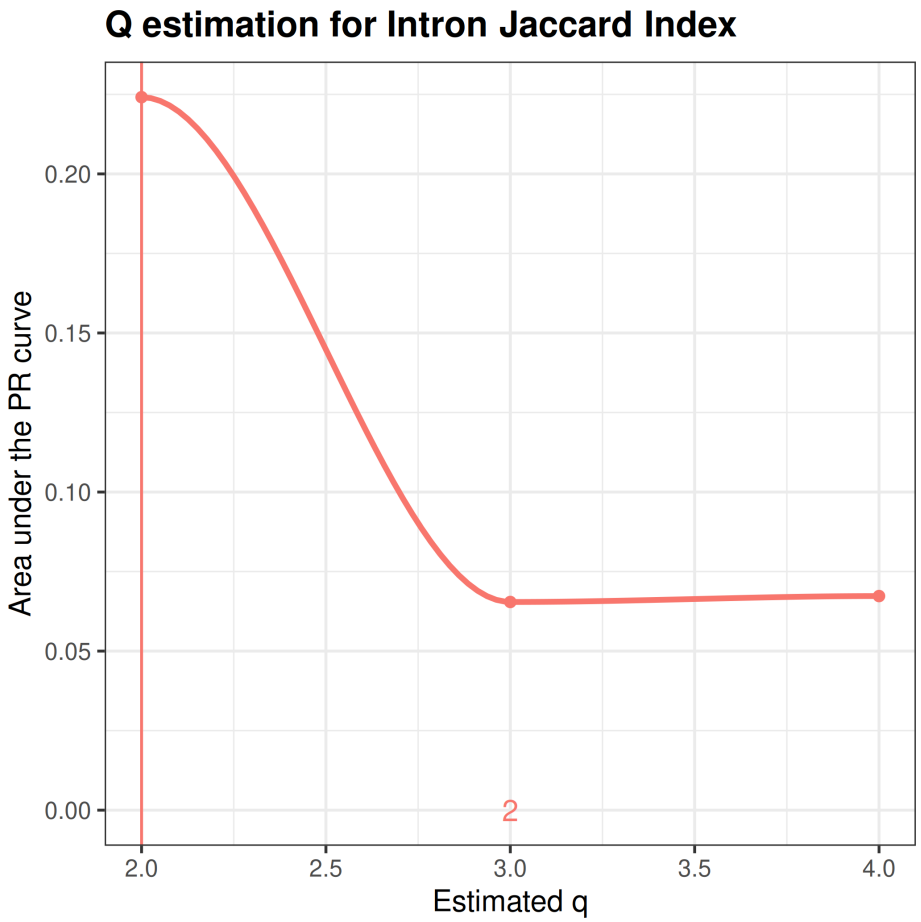
```
set.seed(42)
# Optimal Hard Thresholding (default method)
fds <- estimateBestQ(fds, type="jaccard", plot=FALSE)

## Optimal encoding dimension: 1

plotEncDimSearch(fds, type="jaccard", plotType="sv")
```



```
# Hyperparameter optimization (grid-search)
fds <- estimateBestQ(fds, type="jaccard", useOHT=FALSE, plot=FALSE)
plotEncDimSearch(fds, type="jaccard", plotType="auc")
```




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

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

4.2 P-value calculation

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

$$p_{ij} = 2 \cdot min \left\{ \frac{1}{2}, \sum_{0}^{k_{ij}} BB(k_{ij}, n_{ij}, \mu_{ij}, \rho_i), 1 - \sum_{0}^{k_{ij-1}} BB(k_{ij}, n_{ij}, \mu_{ij}, \rho_i) \right\}, \quad 6$$

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="jaccard")
head(pVals(fds, type="jaccard"))
```

```
##   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[5]. Alternatively, all adjustment methods supported by `p.adjust` can be used via the `method` argument.

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

```
##   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
```

With FRASER 2.0 we introduce the option to limit FDR correction to a subset of genes based on prior knowledge, e.g. genes that contain a rare variant per sample. To use this option, provide a list of genes per sample during FDR computation:

```
genesOfInterest <- list("sample1"=c("XAB2", "PNPLA6", "STXBP2", "ARHGEF18"),
                        "sample2"=c("ARHGEF18", "TRAPPC5"))
fds <- calculatePadjValues(fds, type="jaccard",
              subsets=list("exampleSubset"=genesOfInterest))
head(padjVals(fds, type="jaccard", subsetName="exampleSubset"))
```

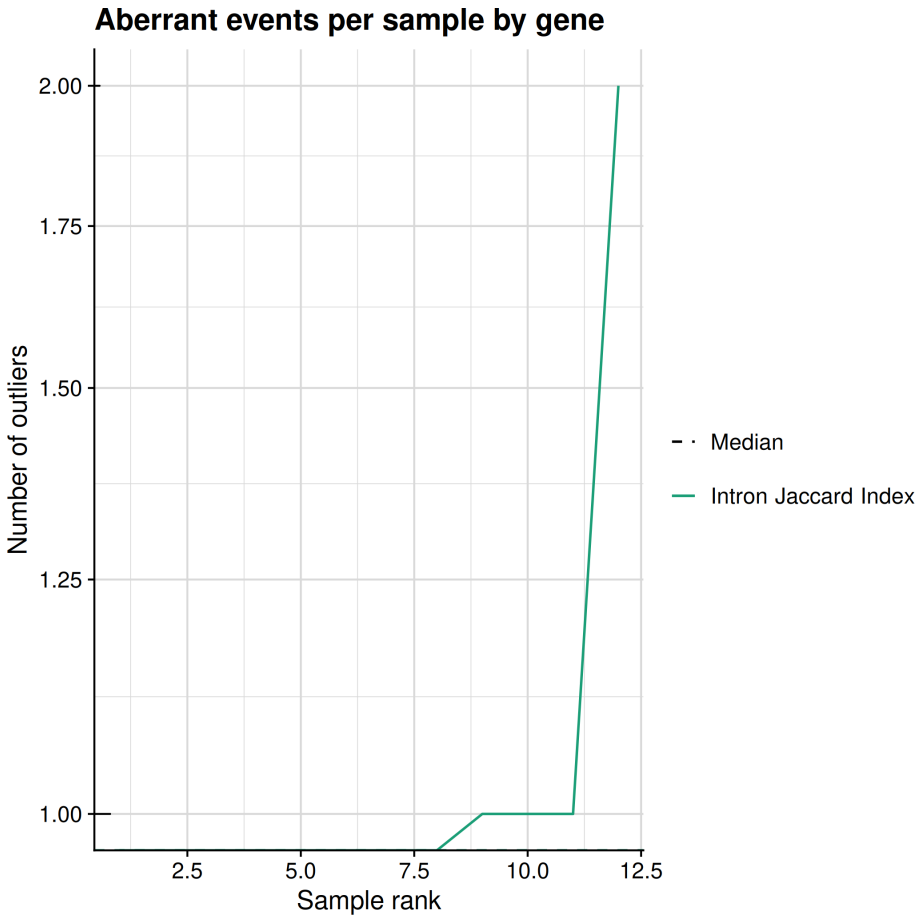
```
##   sample1 sample2 sample3 sample4 sample5 sample6 sample7 sample8 sample9
## 1       1       1      NA      NA      NA      NA      NA      NA      NA
## 2       1       1      NA      NA      NA      NA      NA      NA      NA
## 3       1       1      NA      NA      NA      NA      NA     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

4.3 Result visualization

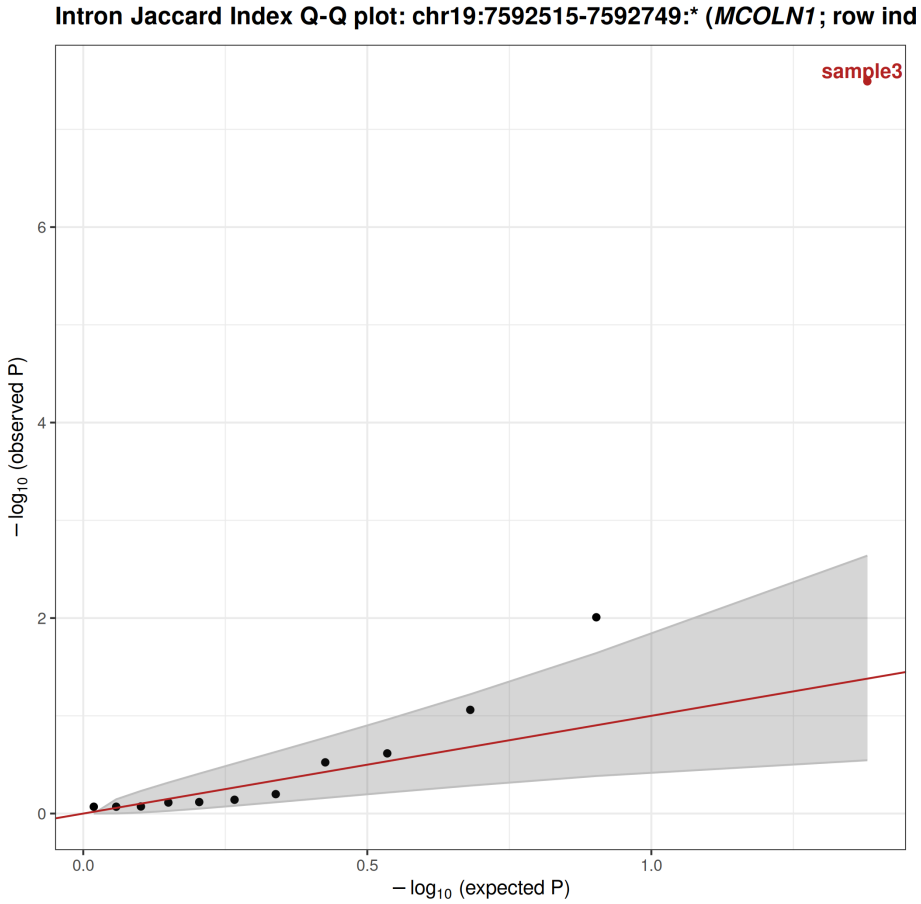
Besides the plotting methods `plotVolcano`, `plotExpression`, `plotExpectedVsObservedPsi`, `plotSpliceMetricRank`, `plotFilterExpression` and `plotEncDimSearch` used above, the *FRASER* package provides additional functions to visualize the results:

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

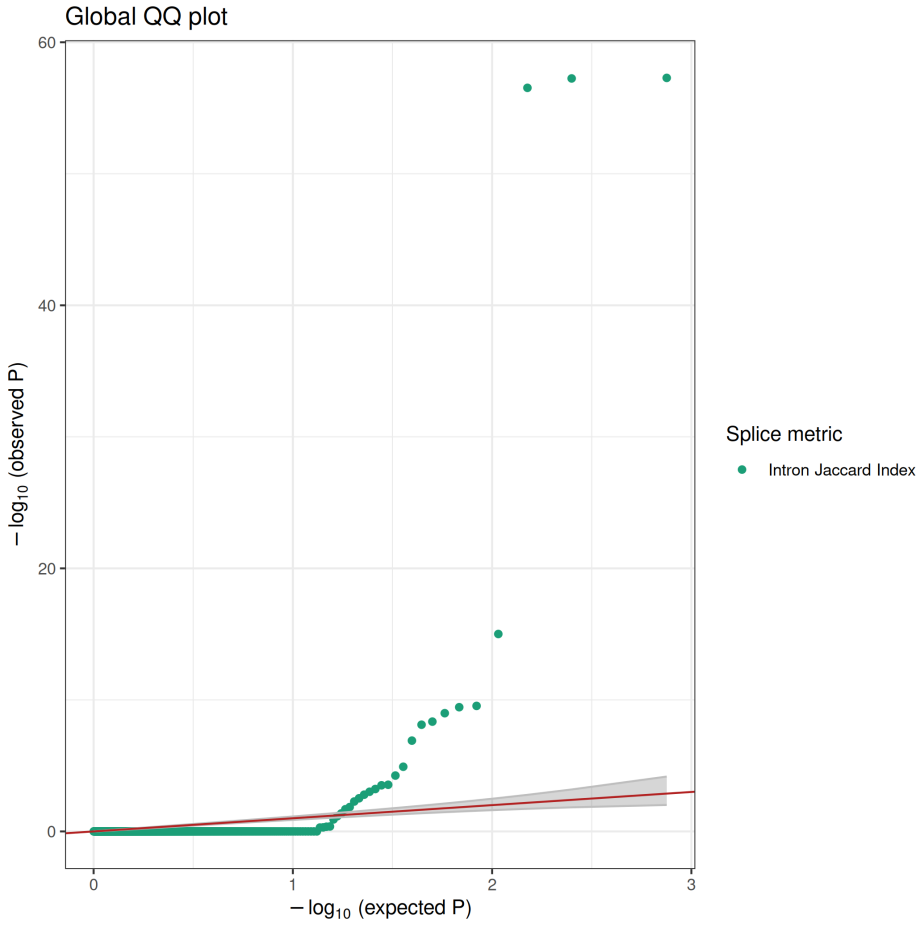
```
plotAberrantPerSample(fds)
```




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

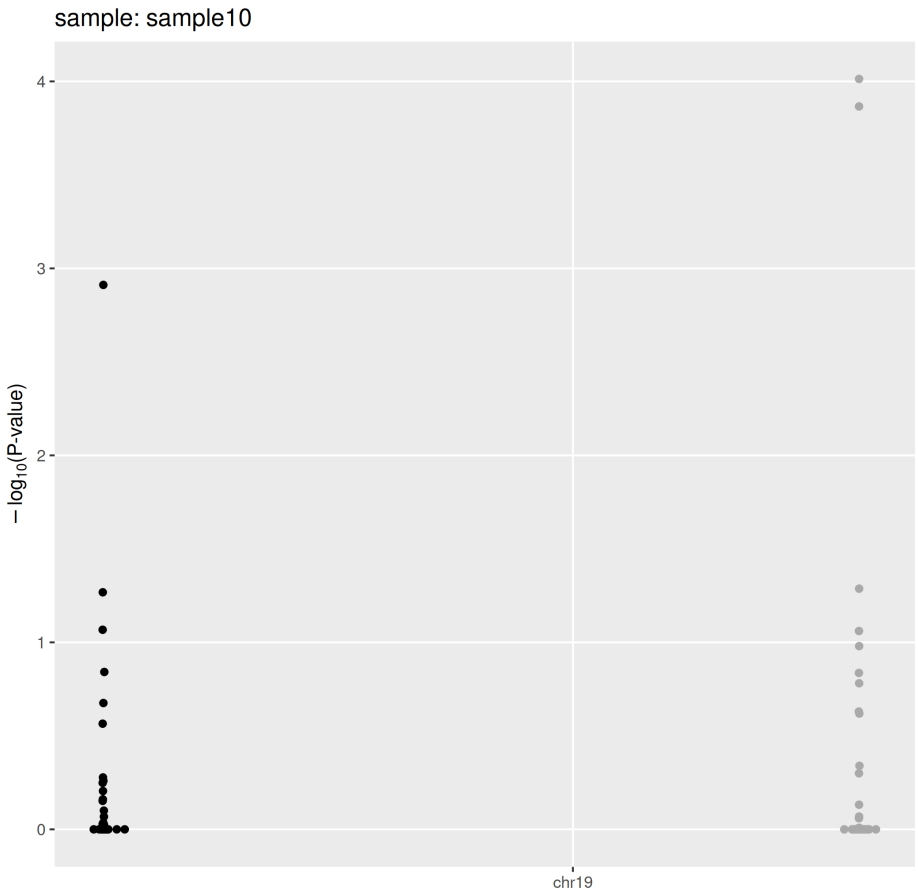



```
# global qq-plot (on gene level since aggregate=TRUE)
plotQQ(fds, aggregate=TRUE, global=TRUE)
```

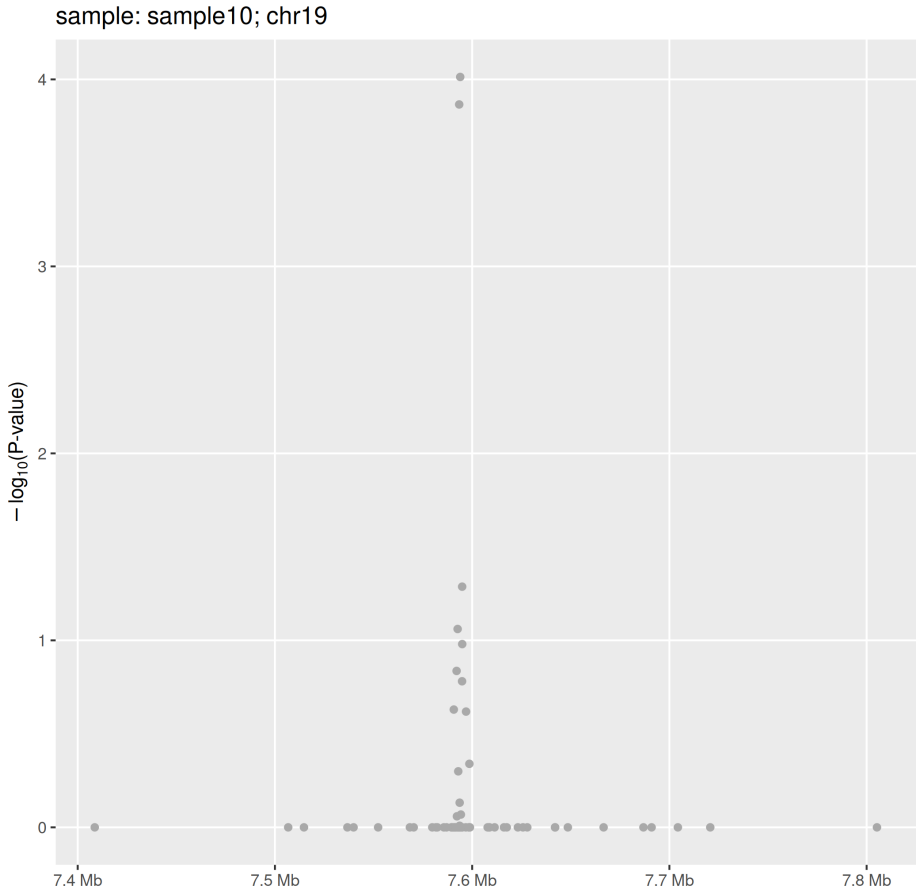


The `plotManhattan` function can be used to visualize the p-values along with the genomic coordinates of the introns:

```
plotManhattan(fds, sampleID="sample10")
```

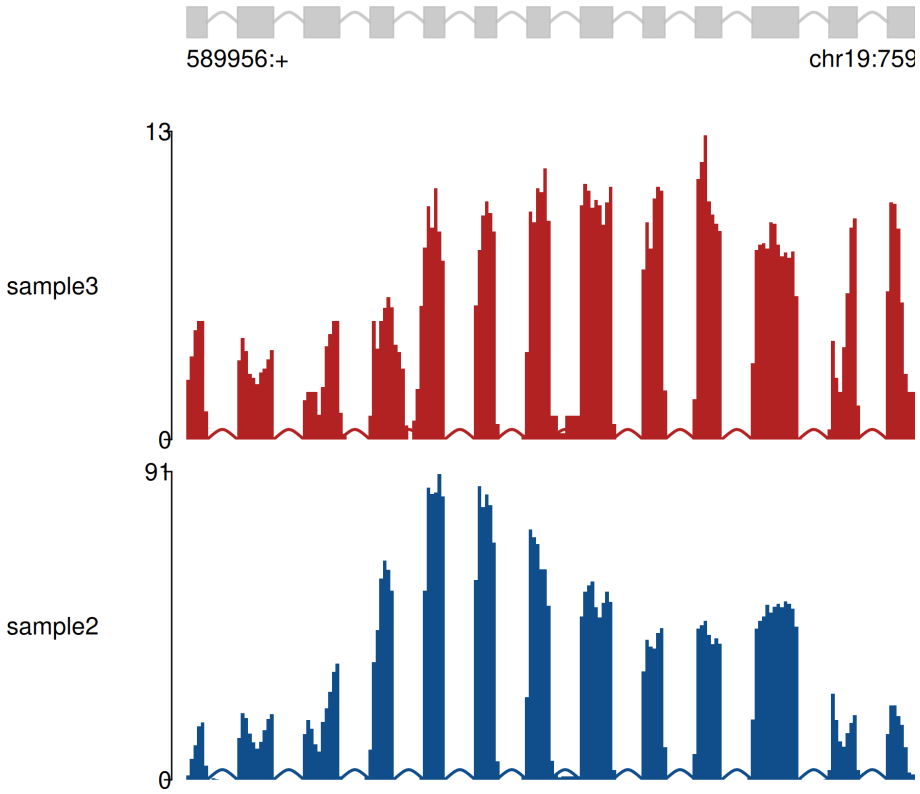



```
plotManhattan(fds, sampleID="sample10", chr="chr19")
```



Finally, when one has access to the bam files from which the split and unsplit counts of FRASER were created, the `plotBamCoverage` and `plotBamCoverageFromResultTable` functions use the *SGSeq* package to allow visualizing the read coverage in the bam file a certain intron from the results table or within a given genomic region as a sashimi plot:

```
### plot coverage from bam file for a certain genomic region
fds <- createTestFraserSettings()
vizRange <- GRanges(seqnames="chr19",
                IRanges(start=7587496, end=7598895),
                strand="+")
plotBamCoverage(fds, gr=vizRange, sampleID="sample3",
                control_samples="sample2", min_junction_count=5,
                curvature_splicegraph=1, curvature_coverage=1,
                mar=c(1, 7, 0.1, 3))
```

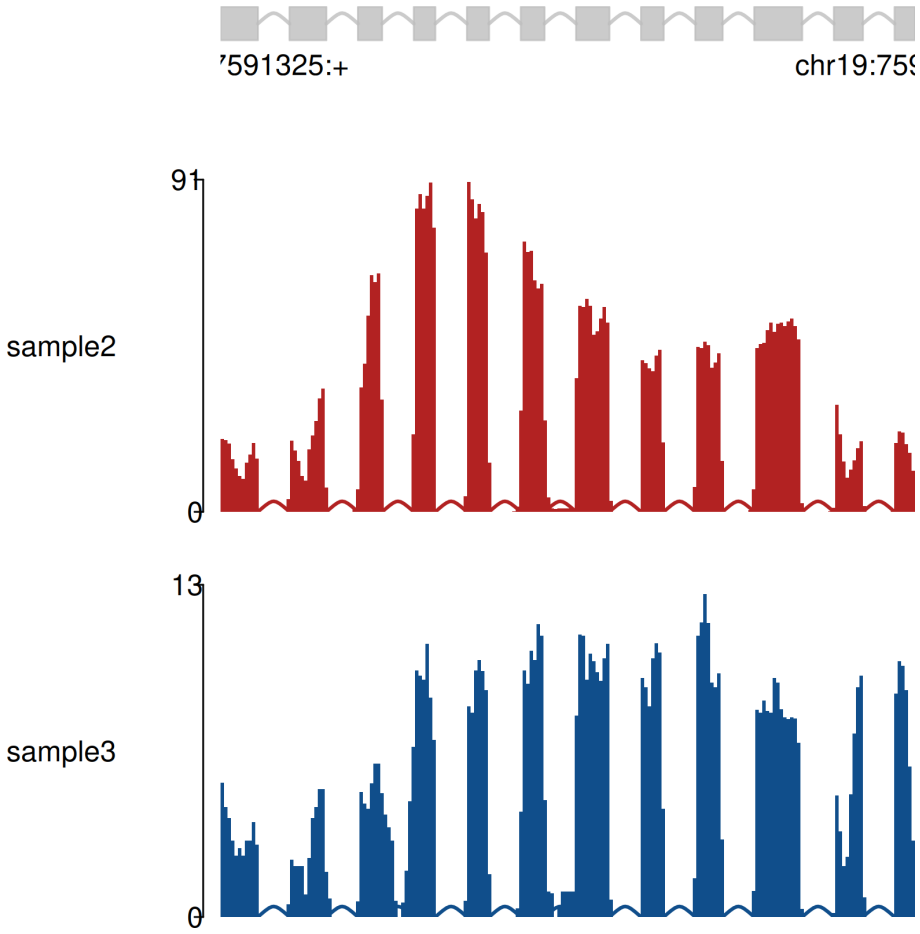



```
### plot coverage from bam file for a row in the result table
fds <- createTestFraserDataSet()

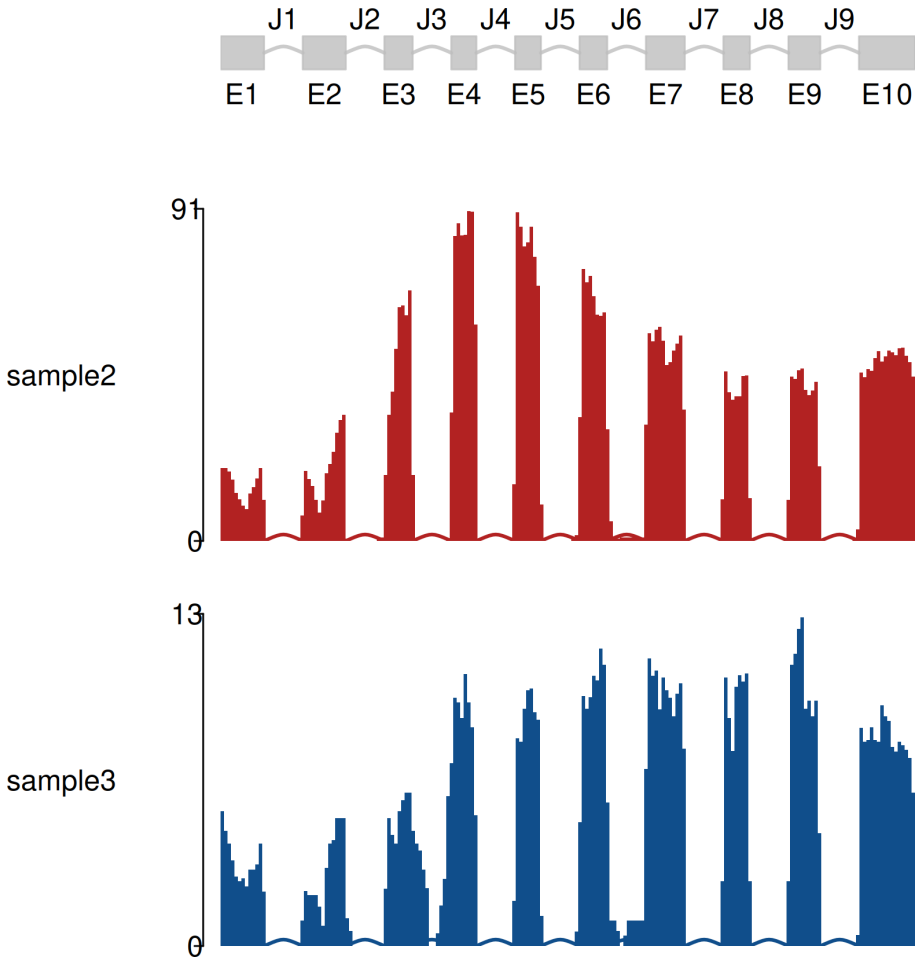
# load gene annotation
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
require(org.Hs.eg.db)
orgDb <- org.Hs.eg.db

# get results table
res <- results(fds, padjCutoff=NA, deltaPsiCutoff=NA)
res_dt <- as.data.table(res)
res_dt <- res_dt[sampleID == "sample2",]

# plot full range of gene highlighting the outlier intron
plotBamCoverageFromResultTable(fds, result=res_dt[1,], show_full_gene=TRUE,
    txdb=txdb, orgDb=orgDb, control_samples="sample3")
```




```
# plot only certain range around the outlier intron
plotBamCoverageFromResultTable(fds, result=res_dt[1,], show_full_gene=FALSE,
     control_samples="sample3", curvature_splicegraph=0.5, txdb=txdb,
     curvature_coverage=0.5, right_extension=5000, left_extension=5000,
     splicegraph_labels="id")
```



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] Matan Gavish and David L. Donoho. The Optimal Hard Threshold for Singular Values is 4/sqrt(3). URL: http://arxiv.org/pdf/1305.5870.

[5] 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 4.6.2 (2026-10-29)
## Platform: x86_64-pc-linux-gnu
## Running under: Ubuntu 26.04.1 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.32.so;  LAPACK version 3.12
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [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
##
## time zone: Etc/UTC
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] data.table_1.18.6.1
##  [2] org.Hs.eg.db_3.23.1
##  [3] TxDb.Hsapiens.UCSC.hg19.knownGene_3.24.0
##  [4] GenomicFeatures_1.65.0
##  [5] AnnotationDbi_1.75.2
##  [6] FRASER_2.9.2
##  [7] SummarizedExperiment_1.43.0
##  [8] Biobase_2.73.2
```

```
##  [9] MatrixGenerics_1.25.0
## [10] matrixStats_1.5.0
## [11] Rsamtools_2.29.0
## [12] Biostrings_2.81.9
## [13] XVector_0.53.0
## [14] GenomicRanges_1.65.4
## [15] IRanges_2.47.5
## [16] S4Vectors_0.51.10
## [17] Seqinfo_1.3.2
## [18] BiocGenerics_0.59.12
## [19] generics_0.1.4
## [20] knitr_1.52
## [21] BiocParallel_1.47.0
##
## loaded via a namespace (and not attached):
##   [1] splines_4.6.2             BiocIO_1.23.3
##   [3] bitops_1.1-0              filelock_1.0.3
##   [5] tibble_3.3.1              R.oo_1.27.1
##   [7] graph_1.91.0              rpart_4.1.27
##   [9] XML_3.99-0.25             lifecycle_1.0.5
##  [11] httr2_1.3.0               OrganismDbi_1.55.1
##  [13] ensembldb_2.37.3          lattice_0.23-1
##  [15] dendextend_1.19.1         backports_1.5.1
##  [17] magrittr_2.0.5            Hmisc_5.3-0
##  [19] plotly_4.12.1             rmarkdown_2.32
##  [21] yaml_2.3.12               otel_0.2.0
##  [23] RUnit_0.4.33.1            ggbio_1.61.0
##  [25] cowplot_1.2.0             DBI_1.3.0
##  [27] buildtools_1.0.0          RColorBrewer_1.1-3
##  [29] abind_1.4-8               purrr_1.2.2
##  [31] R.utils_2.13.0            AnnotationFilter_1.37.0
##  [33] biovizBase_1.61.0         RCurl_1.98-1.20
##  [35] nnet_7.3-21               pracma_2.4.6
##  [37] VariantAnnotation_1.59.4  seriation_1.6.0
##  [39] ggrepel_0.9.8             RMTstat_0.3.2
##  [41] maketools_1.3.2           pheatmap_1.0.13
##  [43] BiocStyle_2.41.0          DelayedMatrixStats_1.35.0
##  [45] codetools_0.2-20          DelayedArray_0.39.8
##  [47] tidyselect_1.2.1          PRROC_1.4
##  [49] UCSC.utils_1.9.0          OUTRIDER_1.31.0
##  [51] farver_2.1.2              viridis_0.6.5
##  [53] TSP_1.2.7                 base64enc_0.1-6
##  [55] BiocFileCache_3.3.1       webshot_0.5.5
##  [57] GenomicAlignments_1.49.2  jsonlite_2.0.0
##  [59] Formula_1.2-6             iterators_1.0.14
```

```
##  [61] foreach_1.5.2             tools_4.6.2
##  [63] progress_1.2.3            Rcpp_1.1.2
##  [65] glue_1.8.1                gridExtra_2.3.1
##  [67] SparseArray_1.13.4        BiocBaseUtils_1.15.1
##  [69] xfun_0.62                 mgcv_1.9-4
##  [71] DESeq2_1.53.7             GenomeInfoDb_1.49.2
##  [73] dplyr_1.2.1               ca_0.72
##  [75] HDF5Array_1.41.3          withr_3.0.3
##  [77] BiocManager_1.30.27       fastmap_1.2.0
##  [79] rhdf5filters_1.25.5       digest_0.6.39
##  [81] R6_2.6.1                  colorspace_2.1-4
##  [83] dichromat_2.0-1           biomaRt_2.69.6
##  [85] RSQLite_3.53.3            cigarillo_1.3.1
##  [87] R.methodsS3_1.8.2         h5mread_1.5.3
##  [89] utf8_1.2.6                tidyr_1.3.2
##  [91] rtracklayer_1.73.0        prettyunits_1.2.0
##  [93] httr_1.4.9                htmlwidgets_1.6.4
##  [95] S4Arrays_1.13.2           pkgconfig_2.0.3
##  [97] gtable_0.3.6              blob_1.3.0
##  [99] registry_0.5-1            S7_0.2.2
## [101] sys_3.4.3                 htmltools_0.5.9
## [103] RBGL_1.89.0               ProtGenerics_1.45.0
## [105] scales_1.4.0              Rsubread_2.27.7
## [107] png_0.1-9                 SGSeq_1.47.0
## [109] rstudioapi_0.19.0         reshape2_1.4.5
## [111] rjson_0.2.23              checkmate_2.3.4
## [113] nlme_3.1-171              curl_8.1.0
## [115] cachem_1.1.0              rhdf5_2.57.20
## [117] stringr_1.6.0             parallel_4.6.2
## [119] foreign_0.8-92            restfulr_0.0.17
## [121] pillar_1.11.1             grid_4.6.2
## [123] vctrs_0.7.3               pcaMethods_2.5.0
## [125] VGAM_1.1-14               dbplyr_2.6.0
## [127] cluster_2.1.8.3           htmlTable_2.5.0
## [129] evaluate_1.0.5            BBmisc_1.13.1
## [131] tinytex_0.61              magick_2.9.1
## [133] cli_3.6.6                 locfit_1.5-9.12
## [135] compiler_4.6.2            rlang_1.3.0
## [137] crayon_1.5.3              heatmaply_1.6.0
## [139] labeling_0.4.3            plyr_1.8.9
## [141] stringi_1.8.9             viridisLite_0.4.3
## [143] assertthat_0.2.1          txdbmaker_1.9.1
## [145] lazyeval_0.2.3            Matrix_1.7-6
## [147] BSgenome_1.81.1           hms_1.1.4
## [149] sparseMatrixStats_1.25.0  bit64_4.8.6
```

```
## [151] ggplot2_4.0.3             Rhdf5lib_2.1.0
## [153] KEGGREST_1.53.6           highr_0.12
## [155] extraDistr_1.10.0.5       igraph_2.3.4
## [157] memoise_2.0.1             bit_4.6.0
```