Package ‘FRASER’

May 3, 2024

Type Package

Title Find RAre Splicing Events in RNA-Seq Data

Version 2.0.0

Date 2024-04-26

Description Detection of rare aberrant splicing events in transcriptome profiles. Read count ratio expectations are modeled by an autoencoder to control for confounding factors in the data. Given these expectations, the ratios are assumed to follow a beta-binomial distribution with a junction specific dispersion. Outlier events are then identified as read-count ratios that deviate significantly from this distribution. FRASER is able to detect alternative splicing, but also intron retention. The package aims to support diagnostics in the field of rare diseases where RNA-seq is performed to identify aberrant splicing defects.

biocViews RNASeq, AlternativeSplicing, Sequencing, Software, Genetics, Coverage

License MIT + file LICENSE

URL https://github.com/gagneurlab/FRASER

BugReports https://github.com/gagneurlab/FRASER/issues

RoxygenNote 7.3.1

Encoding UTF-8

VignetteBuilder knitr

Depends BiocParallel, data.table, Rsamtools, SummarizedExperiment

Imports AnnotationDbi, BBmisc, Biobase, BiocGenerics, biomaRt, BSgenome, cowplot, DelayedArray (>= 0.5.11), DelayedMatrixStats, extraDistr, generics, GenomeInfoDb, GenomicAlignments, GenomicFeatures, GenomicRanges, IRanges, grDevices, ggplot2, ggrepel, HDF5Array, matrixStats, methods, OUTRIDER, pcaMethods, pheatmap, plotly, PRROC, RColorBrewer, rhdf5, Rsubread, R.utils, S4Vectors, stats, tibble, tools, utils, VGAM
Suggests magick, BiocStyle, knitr, rmarkdown, testthat, covr,
TxDb.Hsapiens.UCSC.hg19.knownGene, org.Hs.eg.db, rtracklayer,
SGSeq, ggBio, biovizBase

LinkingTo RcppArmadillo, Rcpp

FraserDataSet-class.R AllGenerics-definitions.R AllGenerics.R
Fraser-pipeline.R annotationOfRanges.R beta-binomial-testing.R
calculatePSIValue.R countRNAseqData.R example_functions.R
filterExpression.R find_encoding_dimensions.R getURLs.R
helper-functions.R mergeExternalData.R saveHDF5Objects.R
plotMethods.R resultAnnotations.R zzz.R

git_url https://git.bioconductor.org/packages/FRASER

git_branch RELEASE_3_19

git_last_commit 43578dd

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19

Date/Publication 2024-05-03

Author Christian Mertes [aut, cre] (<https://orcid.org/0000-0002-1091-205X>),
Ines Scheller [aut] (<https://orcid.org/0000-0003-4533-7857>),
Karoline Lutz [ctb],
Vicente Yepez [aut] (<https://orcid.org/0000-0001-7916-3643>),
Julien Gagneur [aut] (<https://orcid.org/0000-0002-8924-8365>)

Maintainer Christian Mertes <mertes@in.tum.de>

Contents

annotateRanges ................................................. 3
assayNames,FraserDataSet-method ............................. 5
assays,FraserDataSet-method ................................. 5
calculatePSIVales ........................................... 6
countRNA ...................................................... 7
createTestFraserSettings ....................................... 11
filterVariability ............................................. 12
fit .......................................................... 14
FRASER ......................................................... 16
FraserDataSet ................................................ 19
FraserDataSet-class ......................................... 21
getter_setter_functions ....................................... 21
injectOutliers ................................................ 24
K .............................................................. 25
length,FraserDataSet-method ................................. 27
loadFraserDataSet ........................................... 27
makeSimulatedFraserDataSet ................................. 28
annotateRanges

Annotates the given FraserDataSet with the HGNC symbol with bioRmRt

Description
Annotates the given FraserDataSet with the HGNC symbol with bioRmRt

Usage
annotateRanges(
  fds,
  feature = "hgnc_symbol",
  featureName = feature,
  biotype = list("protein_coding"),
  ensembl = NULL,
  GRCh = 37
)

annotateRangesWithTxDb(
  fds,
  feature = "SYMBOL",
  featureName = "hgnc_symbol",
  keytype = "ENTREZID",
  txdb = NULL,
  orgDb = NULL,
  filter = list()
)

Arguments
fds FraserDataSet
feature Defines which feature (default is HGNC symbol) should be annotated. Has to be the bioRmRt feature name or a column name in orgDb.
featureName The column name of the feature in the FraserDataSet mcols.
biotype The biotype for bioRmRt.
ensembl: The ensembl that should be used. If NULL, the default one is used (hsapiens_gene_ensembl, GRCh37).

GRCh: GRCh version to connect to. If this is NULL, then the current GRCh38 is used. Otherwise, this can only be 37 (default) at the moment (see useEnsembl).

keytype: The keytype or column name of gene IDs in the TxDb object (see keytypes for a list of available ID types).

txdb: A TxDb object. If this is NULL, then the default one is used, currently this is TxDb.Hsapiens.UCSC.hg19.knownGene.

orgDb: An orgDb object or a data table to map the feature names. If this is NULL, then org.Hs.eg.db is used as the default.

filter: A named list specifying the filters which should be applied to subset to e.g. only protein-coding genes for annotation. names(filter) needs to be column names in the given orgDb object (default: no filtering).

Value

FraserDataSet

Examples

fds <- createTestFraserDataSet()

### Two ways to annotate ranges with gene names:  # either using biomart with GRCh38
try(
  fds <- annotateRanges(fds, GRCh=38)
  rowRanges(fds, type="j")[,c("hgnc_symbol")]
)

# either using biomart with GRCh37
try(
  fds <- annotateRanges(fds, featureName="hgnc_symbol_37", GRCh=37)
  rowRanges(fds, type="j")[,c("hgnc_symbol_37")]
)

# or with a provided TxDb object
require(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
require(org.Hs.eg.db)
orgDb <- org.Hs.eg.db
fds <- annotateRangesWithTxDb(fds, txdb=txdb, orgDb=orgDb)
rowRanges(fds, type="j")[,"hgnc_symbol"]
### assayNames, FraserDataSet-method

Returns the assayNames of FRASER

**Description**

Returns the assayNames of FRASER

**Usage**

```r
## S4 method for signature 'FraserDataSet'
assayNames(x)
```

**Arguments**

- `x` FraserDataSet

**Value**

Character vector

---

### assays, FraserDataSet-method

Returns the assay for the given name/index of the FraserDataSet

**Description**

Returns the assay for the given name/index of the FraserDataSet

**Usage**

```r
## S4 method for signature 'FraserDataSet'
assays(x, withDimnames = TRUE, ...)  

## S4 replacement method for signature 'FraserDataSet,SimpleList'
assays(x, withDimnames = TRUE, HDF5 = TRUE, type = NULL, ...) <- value  

## S4 replacement method for signature 'FraserDataSet,list'
assays(x, withDimnames = TRUE, HDF5 = TRUE, type = NULL, ...) <- value  

## S4 replacement method for signature 'FraserDataSet,DelayedMatrix'
assays(x, withDimnames = TRUE, HDF5 = TRUE, type = NULL, ...) <- value
```
calculatePSIVValues

Arguments

x FraserDataSet
withDimnames Passed on to SummarizedExperiment::assays()
... Parameters passed on to SummarizedExperiment::assays()
HDF5 Logical value indicating whether the assay should be stored as a HDF5 file.
type The psi type.
value The new value to which the assay should be set.

Value

(Delayed) matrix.

Description

This function calculates the PSI values for each junction and splice site based on the FraserDataSet object.

Usage

calculatePSIVValues(
  fds,
  types = psiTypes,
  overwriteCts = FALSE,
  BPPARAM = bpparam()
)

Arguments

fds A FraserDataSet object

Value

FraserDataSet
**Examples**

```r
fds <- createTestFraserDataSet()
fds <- calculatePSIValues(fds, types="jaccard")

### usually one would run this function for all psi types by using:
# fds <- calculatePSIValues(fds)
```

---

**countRNA**

*Count RNA-seq data*

**Description**

The FRASER package provides multiple functions to extract and count both split and non-spliced reads from bam files. See Detail and Functions for more information.

**Usage**

```r
countRNAData(
  fds,
  NcpuPerSample = 1,
  minAnchor = 5,
  recount = FALSE,
  BPPARAM = bpparam(),
  genome = NULL,
  junctionMap = NULL,
  filter = TRUE,
  minExpressionInOneSample = 20,
  keepNonStandardChromosomes = TRUE,
  countDir = file.path(workingDir(fds), "savedObjects", nameNoSpace(name(fds))),
  ...
)
```

```r
getSplitReadCountsForAllSamples(
  fds,
  NcpuPerSample = 1,
  junctionMap = NULL,
  recount = FALSE,
  BPPARAM = bpparam(),
  genome = NULL,
  countFiles = NULL,
  keepNonStandardChromosomes = TRUE,
  outDir = file.path(workingDir(fds), "savedObjects", nameNoSpace(name(fds)), "splitCounts")
)
```

```r
getNonSplitReadCountsForAllSamples(
  fds,
```
splitCountRanges,
NcpuPerSample = 1,
minAnchor = 5,
recount = FALSE,
BPPARAM = bpparam(),
longRead = FALSE,
outDir = file.path(workingDir(fds), "savedObjects", nameNoSpace(name(fds)),
    "nonSplitCounts")
)

addCountsToFraserDataSet(fds, splitCounts, nonSplitCounts)

countSplitReads(
    sampleID,
    fds,
    NcpuPerSample = 1,
genome = NULL,
    recount = FALSE,
    keepNonStandardChromosomes = TRUE,
bamfile = bamFile(fds[, sampleID]),
pairedend = pairedEnd(fds[, sampleID]),
strandmode = strandSpecific(fds[, sampleID]),
cacheFile = getSplitCountCacheFile(sampleID, fds),
scanbamparam = scanBamParam(fds),
coldata = colData(fds)
)

mergeCounts(
    countList,
    fds,
junctionMap = NULL,
    assumeEqual = FALSE,
spliceSiteCoords = NULL,
BPPARAM = SerialParam()
)

countNonSplicedReads(
    sampleID,
    splitCountRanges,
    fds,
    NcpuPerSample = 1,
minAnchor = 5,
recount = FALSE,
spliceSiteCoords = NULL,
longRead = FALSE
)
countRNA

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fds</td>
<td>A FraserDataSet object</td>
</tr>
<tr>
<td>NcpuPerSample</td>
<td>A BiocParallel param object or a positive integer to configure the parallel backend of the internal loop per sample</td>
</tr>
<tr>
<td>minAnchor</td>
<td>Minimum overlap around the Donor/Acceptor for non spliced reads. Default to 5</td>
</tr>
<tr>
<td>recount</td>
<td>if TRUE the cache is ignored and the bam file is recounted.</td>
</tr>
<tr>
<td>BPARAM</td>
<td>the BiocParallel parameters for the parallelization</td>
</tr>
<tr>
<td>genome</td>
<td>NULL (default) or a character vector specifying the names of the reference genomes that were used to align the reads for each sample. The names have to be in a way accepted by the getBSgenome function. Available genomes can be listed using the available.genomes function from the BSgenome package. If genome is of length 1, the same reference genome will be used for all samples. If genome is supplied and strandSpecific(fds) == 0L (unstranded), then the strand information will be estimated by checking the dinucleotides found at the intron boundaries (see summarizeJunctions in GenomicAlignments package for details). This can e.g. help to avoid ambiguities when adding gene names from a gene annotation to the introns in a later step.</td>
</tr>
<tr>
<td>junctionMap</td>
<td>A object or file containing a map of all junctions of interest across all samples</td>
</tr>
<tr>
<td>filter</td>
<td>If TRUE, splice sites of introns with low read support in all samples are not considered when calculating the non-split reads. This helps to speed up the subsequent steps.</td>
</tr>
<tr>
<td>minExpressionInOneSample</td>
<td>The minimal split read count in at least one sample that is required for an intron to pass the filter.</td>
</tr>
<tr>
<td>keepNonStandardChromosomes</td>
<td>Logical value indicating if non standard chromosomes should also be counted. Defaults to TRUE.</td>
</tr>
<tr>
<td>countDir</td>
<td>The directory in which the tsv containing the position and counts of the junctions should be placed.</td>
</tr>
<tr>
<td>...</td>
<td>Further parameters passed on to Rsubread::featureCounts.</td>
</tr>
<tr>
<td>countFiles</td>
<td>If specified, the split read counts for all samples are read from the specified files. Should be a vector of paths to files containing the split read counts for the individual samples. Reading from files is only supported for tsv(.gz) or RDS files containing GRranges objects. The order of the individual sample files should correspond to the order of the samples in the fds.</td>
</tr>
<tr>
<td>outDir</td>
<td>The full path to the output folder containing the merged counts. If the given folder already exists and stores a SummarizedExperiment object, the counts from this folder will be read in and used in the following (i.e. the reads are not recounted), unless the option recount=TRUE is used. If this folder doesn’t exist or if recount=TRUE, then it will be created after counting has finished.</td>
</tr>
<tr>
<td>splitCountRanges</td>
<td>The merged GRanges object containing the positions of all the introns in the dataset over all samples.</td>
</tr>
</tbody>
</table>
longRead If TRUE, then the isLongRead option of Rsubread::featureCounts is used when counting the non spliced reads overlapping splice sites.

splitCounts The SummarizedExperiment object containing the position and counts of all the introns in the dataset for all samples.

nonSplitCounts The SummarizedExperiment object containing the position and non split read counts of all splice sites present in the dataset for all samples.

sampleID The ID of the sample to be counted.

bamfile The BAM file to be used to extract the counts. Defaults to the BAM file defined in the FraserDataSet object.

pairedend TRUE or FALSE if the BAM file is paired end. Defaults to the value specified in the FraserDataSet object.

strandmode 0 (no, default), 1 (stranded), or 2 (revers) to specify the used protocol for the RNA-seq experiment.

cacheFile File path to the cache, where counts are stored.

scanbamparam The ScanBamParam object which is used for loading the reads from the BAM file before counting. Defaults to the params stored in the FraserDataSet object.

coldata The colData as given by the FraserDataSet object.

countList A list of GRanges objects containing the counts that should be merged into one object.

assumeEqual Logical indicating whether all objects in countList can be assumed to contain counts for the same ranges. If FALSE, merging of the ranges is performed.

spliceSiteCoords A GRanges object containing the positions of the splice sites. If it is NULL, then splice sites coordinates are calculated first based on the positions of the junctions defined from the split reads.

Details

The functions described in this file extract and count both the split and the non-spliced reads from bam files.

countRNAData is the main function that takes care of all counting steps and returns a FraserDataSet containing the counts for all samples in the fds.

getAddressSplitReadCountsForAllSamples counts split reads for all samples and getNonSplitReadCountsForAllSamples counts non split reads overlapping splice sites for all samples. addCountsToFrasierDataSet adds these counts to an existing fds.

countSplitReads calculates the split read counts for a single sample. countNonSplicedReads counts the non split reads overlapping with splice sites for a single sample.

mergeCounts merges the counts from different samples into a single count object, where the counts for junctions that are not present in a sample are set to zero.

Value

countRNAData returns a FraserDataSet.

getAddressSplitReadCountsForAllSamples returns a GRanges object.
createTestFraserSettings

**getNonSplitReadCountsForAllSamples** returns a GRanges object.
**addCountsToFraserDataSet** returns a FraserDataSet.
**countSplitReads** returns a GRanges object.
**mergeCounts** returns a SummarizedExperiment object.
**countNonSplicedReads** returns a GRanges object.

### Functions

- **countRNAData()**: This method extracts and counts the split reads and non spliced reads from RNA bam files.
- **getSplitReadCountsForAllSamples()**: This method creates a GRanges object containing the split read counts from all specified samples.
- **getNonSplitReadCountsForAllSamples()**: This method creates a GRanges object containing the non split read counts at the exon-intron boundaries inferred from the GRanges object containing the positions of all the introns in this dataset.
- **addCountsToFraserDataSet()**: This method adds the split read and non split read counts to a existing FraserDataSet containing the settings.
- **countSplitReads()**: This method counts all split reads in a bam file for a single sample.
- **mergeCounts()**: This method merges counts for multiple samples into one SummarizedExperiment object.
- **countNonSplicedReads()**: This method counts non spliced reads based on the given target (acceptor/donor) regions for a single sample.

### Examples

```r
# On Windows SNOW is the default for the parallele backend, which can be # very slow for many but small tasks. Therefore, we will use # for the example the SerialParam() backend.
if(.Platform$OS.type != "unix") {
  register(SerialParam())
}

fds <- countRNAData(createTestFraserSettings())
```

---

**createTestFraserSettings**

*Create a test dataset*

**Description**

Create a test case dataset based on the test sample annotation to be used in the vignette and to explore the functionality of the FRASER package. Dependent on the request only the sample annotation or a full fitted model is returned.
Usage

createTestFraserSettings(workingDir = "FRASER_output")

createTestFraserDataSet(
    workingDir = "FRASER_output",
    rerun = FALSE,
    metrics = "jaccard"
)

Arguments

workingDir  Directory where to store HDF5 and RDS files. Defaults to FRASER_output in the current working directory.
rerun  Defaults to FALSE. If set to TRUE it reruns the full fit of the model.
metrics  The splice metrics that should be included in the test fds. One or several of 'jaccard', 'psi5', 'psi3' or 'theta'.

Value

A FraserDataSet object that contains a test case

Examples

fds <- createTestFraserSettings()
fds

fds <- createTestFraserDataSet()
fds

filterVariability

Filtering FraserDataSets

Description

This method can be used to filter out introns that are not reliably detected and to remove introns with no variability between samples.

Usage

filterVariability(object, ...)

filterExpressionAndVariability(
    object,
    minExpressionInOneSample = 20,
    quantile = 0.75,
    quantileMinExpression = 10,
filterVariability

minDeltaPsi = 0,
filter = TRUE,
delayed = ifelse(ncol(object) <= 300, FALSE, TRUE),
filterOnJaccard = TRUE,
BPPARAM = bpparam()
)

## S4 method for signature 'FraserDataSet'
filterExpression(
  object,
  minExpressionInOneSample = 20,
  quantile = 0.75,
  quantileMinExpression = 10,
  filter = TRUE,
  delayed = ifelse(ncol(object) <= 300, FALSE, TRUE),
  filterOnJaccard = TRUE,
  BPPARAM = bpparam()
)

## S4 method for signature 'FraserDataSet'
filterVariability(
  object,
  minDeltaPsi = 0,
  filter = TRUE,
  delayed = ifelse(ncol(object) <= 300, FALSE, TRUE),
  filterOnJaccard = TRUE,
  BPPARAM = bpparam()
)

Arguments

object A FraserDataSet object
...
Further parameters passed on to Rsubread::featureCounts.

minExpressionInOneSample
  The minimal read count in at least one sample that is required for an intron to pass the filter.

quantile
  Defines which quantile should be considered for the filter.

quantileMinExpression
  The minimum read count an intron needs to have at the specified quantile to pass the filter.

minDeltaPsi
  Only introns for which the maximal difference in the psi value of a sample to the mean psi of the intron is larger than this value pass the filter.

filter
  If TRUE, a subsetted fds containing only the introns that passed all filters is returned. If FALSE, no subsetting is done and the information of whether an intron passed the filters is only stored in the mcols.

delayed
  If FALSE, count matrices will be loaded into memory, otherwise the function works on the delayedMatrix representations. The default value depends on the number of samples in the fds-object.
filterOnJaccard

If TRUE, the Intron Jaccard Metric is used to define express introns during filtering. Otherwise, the psi5, psi3 and theta metrics are used (default: TRUE).

BPPARAM

the BiocParallel parameters for the parallelization

Value

A FraserDataSet with information about which junctions passed the filters. If filter=TRUE, the filtered FraserDataSet is returned.

Functions

• filterExpressionAndVariability(): This function filters out both introns with low read support and introns that are not variable across samples.
• filterExpression(FraserDataSet): This function filters out introns and corresponding splice sites that have low read support in all samples.
• filterVariability(FraserDataSet): This function filters out introns and corresponding splice sites that have low read support in all samples.

Examples

fds <- createTestFraserDataSet()
fds <- filterExpressionAndVariability(fds, minDeltaPsi=0.1, filter=FALSE)
mcols(fds, type="jaccard")[, c("maxCount", "passedExpression", "maxDJaccard", "passedVariability")]

plotFilterExpression(fds)
plotFilterVariability(fds)

fit

Fitting the denoising autoencoder

Description

This method corrects for confounders in the data and fits a beta-binomial distribution to the introns/splice sites.

For more details please see FRASER.

Usage

## S3 method for class 'FraserDataSet'
fit(
  object,
  q,
)
type = psiTypes,
rhoRange = c(-30, 30),
weighted = FALSE,
noiseAlpha = 1,
convergence = 1e-05,
iterations = 15,
initialize = TRUE,
control = list(),
BPPARAM = bpparam(),
nSubset = 15000,
minDeltaPsi = 0.1,
...)

Arguments

object
A FraserDataSet object

implementation
The method that should be used to correct for confounders.

q
The encoding dimensions to be used during the fitting procedure. Should be
fitted using optimHyperParams if unknown. If a named vector is provided it is
used for the different splicing types.

type
The type of PSI (jaccard, psi5, psi3 or theta for theta/splicing efficiency)

rhoRange
Defines the range of values that rho parameter from the beta-binomial distribu-
tion is allowed to take. For very small values of rho, the loss can be instable, so
it is not recommended to allow rho < 1e-8.

weighted
If TRUE, the weighted implementation of the autoencoder is used

noiseAlpha
Controls the amount of noise that is added for the denoising autoencoder.

convergence
The fit is considered to have converged if the difference between the previous
and the current loss is smaller than this threshold.

iterations
The maximal number of iterations. When the autoencoder has not yet converged
after these number of iterations, the fit stops anyway.

initialize
If FALSE and a fit has been previously run, the values from the previous fit will
be used as initial values. If TRUE, (re-)initialization will be done.

control
List of control parameters passed on to optim().

BPPARAM
the BiocParallel parameters for the parallelization

nSubset
The size of the subset to be used in fitting if subsetting is used.

minDeltaPsi
Minimal delta psi of an intron to be be considered a variable intron.

... Currently not used

Value

FraserDataSet

See Also

FRASER
Description

This help page describes the FRASER function which can be used run the default FRASER pipeline. This pipeline combines the beta-binomial fit, the computation of Z scores and p values as well as the computation of delta-PSI values.

Usage

```r
FRASER(
  fds,
  q,
  type = fitMetrics(fds),
  implementation = c("PCA", "PCA-BB-Decoder", "AE-weighted", "AE", "BB"),
  iterations = 15,
  BPPARAM = bpparam(),
  correction,
  subsets = NULL,
  ...
)
```

calculateZscore(fds, type = currentType(fds), logit = TRUE)

calculatePvalues(
  fds,
  type = currentType(fds),
  implementation = "PCA",
  BPPARAM = bpparam(),
  distributions = c("betabinomial"),
  capN = 5 * 1e+05
)

calculatePadjValues(
  fds,
  type = currentType(fds),
  method = "BY",
  rhoCutoff = NA,
  geneLevel = TRUE,
  geneColumn = "hgnc_symbol",
  subsets = NULL,
  BPPARAM = bpparam()
)

calculatePadjValuesOnSubset(
  fds,
  ...}
genesToTest,
subsetName,
type = currentType(fds),
method = "BY",
geneColumn = "hgnc_symbol",
BPPARAM = bpparam()
)

Arguments

fds
A FraserDataSet object

q
The encoding dimensions to be used during the fitting procedure. Should be
fitted using optimHyperParams if unknown. If a named vector is provided it is
used for the different splicing types.

type
The type of PSI (jaccard, psi5, psi3 or theta for theta/splicing efficiency)

implementation
The method that should be used to correct for confounders.

iterations
The maximal number of iterations. When the autoencoder has not yet converged
after these number of iterations, the fit stops anyway.

BPPARAM
A BiocParallel object to run the computation in parallel

correction
Deprecated. The name changed to implementation.

subsets
A named list of named lists specifying any number of gene subsets (can differ
per sample). For each subset, FDR correction will be limited to genes in the
subset, and the FDR corrected pvalues stored as an assay in the fds object in
addition to the transcriptome-wide FDR corrected pvalues. See the examples
for how to use this argument.

... Additional parameters passed on to the internal fit function

logit
Indicates if z scores are computed on the logit scale (default) or in the natural
(ψ) scale.

distributions
The distribution based on which the p-values are calculated. Possible are beta-
binomial, binomial and normal.

capN
Counts are capped at this value to speed up the p-value calculation

method
The p.adjust method that should be used for genome-wide multiple testing cor-
rection.

rhoCutoff
The cutoff value on the fitted rho value (overdispersion parameter of the betabi-
nomial) above which junctions are masked with NA during p value adjustment
(default: NA, no masking).

geneLevel
Logical value indicating whether gene-level p values should be calculated. De-
faults to TRUE.

geneColumn
The column name of the column that has the gene annotation that will be used
for gene-level pvalue computation.

genesToTest
A named list with the subset of genes to test per sample. The names must corre-
spond to the sampleIDs in the given fds object.

subsetName
The name under which the resulting FDR corrected pvalues will be stored in
metadata(fds).
Details

All computed values are returned as an FraserDataSet object. To have more control over each analysis step, one can call each function separately.

- `fit` to control for confounding effects and fit the beta binomial model parameters
- `calculatePvalues` to calculate the nominal p values
- `calculatePadjValues` to calculate adjusted p values (per sample)
- `calculateZscore` to calculate the Z scores

Available methods to correct for the confounders are currently: a denoising autoencoder with a BB loss ("AE" and "AE-weighted"), PCA ("PCA"), a hybrid approach where PCA is used to fit the latent space and then the decoder of the autoencoder is fit using the BB loss ("PCA-BB-Decoder"). Although not recommended, it is also possible to directly fit the BB distribution to the raw counts ("BB").

Value

FraserDataSet

Functions

- `FRASER()`: This function runs the default FRASER pipeline combining the beta-binomial fit, the computation of Z scores and p values as well as the computation of delta-PSI values.
- `calculateZscore()`: This function calculates z-scores based on the observed and expected logit psi.
- `calculatePvalues()`: This function calculates two-sided p-values based on the beta-binomial distribution (or binomial or normal if desired). The returned p values are not yet adjusted with Holm’s method per donor or acceptor site, respectively.
- `calculatePadjValues()`: This function adjusts the previously calculated p-values per sample for multiple testing. First, the previously calculated junction-level p values are adjusted with Holm’s method per donor or acceptor site, respectively. Then, if gene symbols have been annotated to junctions (and not otherwise requested), gene-level p values are computed.
- `calculatePadjValuesOnSubset()`: This function does FDR correction only for all junctions in a certain subset of genes which can differ per sample. Requires gene symbols to have been annotated to junctions. As with the full FDR correction across all junctions, first the previously calculated junction-level p values are adjusted with Holm’s method per donor or acceptor site, respectively. Then, gene-level p values are computed.

Author(s)

Christian Mertes <mertes@in.tum.de>
Ines Scheller <scheller@in.tum.de>

See Also

fit
Examples

# set default parallel backend
register(SerialParam())

# preprocessing
fds <- createTestFraserDataSet()

# filtering not expressed introns
fds <- calculatePSIValues(fds)
fds <- filterExpressionAndVariability(fds)

# Run the full analysis pipeline: fits distribution and computes p values
fds <- FRASER(fds, q=2, implementation="PCA")

# afterwards, the fitted fds-object can be saved and results can
# be extracted and visualized, see ?saveFraserDataSet, ?results and
# ?plotVolcano

# The functions run inside the FRASER function can also be directly
# run themselves.
# To directly run the fit function:
fds <- fit(fds, implementation="PCA", q=2, type="jaccard")

# To directly run the nominal and adjusted p value and z score
# calculation, the following functions can be used:
fds <- calculatePValues(fds, type="jaccard")
head(pVals(fds, type="jaccard"))
fds <- calculatePadjValues(fds, type="jaccard", method="BY")
head(padjVals(fds, type="jaccard"))
fds <- calculateZscore(fds, type="jaccard")
head(zScores(fds, type="jaccard"))

# example of restricting FDR correction to subsets of genes of interest
genesOfInterest <- list("sample1"=c("TIMMDC1"), "sample2"=c("MCOLN1"))
fds <- calculatePadjValues(fds, type="jaccard",
    subsets=list("exampleSubset"=genesOfInterest))
padjVals(fds, type="jaccard", subsetName="exampleSubset")
padjVals(fds, type="jaccard", level="gene", subsetName="exampleSubset")
fds <- calculatePadjValues(fds, type="jaccard",
    subsets=list("anotherExampleSubset"=c("TIMMDC1")))
padjVals(fds, type="jaccard", subsetName="anotherExampleSubset")

# only adding FDR corrected pvalues on a subset without calculating
# transcriptome-wide FDR again:
fds <- calculatePadjValuesOnSubset(fds, genesToTest=genesOfInterest, 
    subsetName="setOfInterest", type="jaccard")
padjVals(fds, type="jaccard", subsetName="setOfInterest")
Description

Constructs an FRASER object based on the given input. It can take only the annotation (colData) or count tables (junctions/spliceSites).

Usage

FraserDataSet(colData = NULL, junctions = NULL, spliceSites = NULL, ...)

Arguments

colData  A DataFrame containing the annotation of the samples
junctions, spliceSites  A data.frame like object containing the raw counts for each junction or splice site. It requires the columns startID and endID for the junctions and spliceSiteID and type for the splice sites. Those columns identifies the corresponding splice site for the given junction and map to the splice site. For each sample the counts are saved in a corresponding column with the same name. It can also be a GRanges object.

...  Any parameters corresponding to the slots and their possible values. See FraserDataSet

Value

A FraserDataSet object.

Author(s)

Christian Mertes <mertes@in.tum.de>

Examples

fraser <- FraserDataSet()

# example sample annoation
sampleTable <- fread(system.file("extdata",
  "sampleTable_countTable.tsv", package="FRASER", mustWork=TRUE))

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

# create FRASER object
fds <- FraserDataSet(colData=sampleTable, junctions=junctionCts, spliceSites=spliceSiteCts, name="Example Dataset")
FraserDataSet-class

Description

This class is designed to store the whole FRASER data set needed for an analysis of a disease cohort.

Author(s)

Christian Mertes <mertes@in.tum.de>

getter_setter_functions

Description

This is a collection of small accessor/setter functions for easy access to the values within the FRASER model.

Usage

featureExclusionMask(fds, type = currentType(fds))

featureExclusionMask(fds, type = currentType(fds)) <- value

rho(fds, type = currentType(fds))

zScores(fds, type = currentType(fds), byGroup = FALSE, ...)

pVals(
  fds,
  type = currentType(fds),
  level = "site",
  filters = list(),
  dist = "BetaBinomial",
  ...
)

padjVals(
  fds,
  type = currentType(fds),
  dist = c("BetaBinomial"),
  level = "site",
  subsetName = NULL,
getter_setter_functions

```r
filters = list(),
...
)

availableFDRsubsets(fds)
predictedMeans(fds, type = currentType(fds))
deltaPsiValue(fds, type = currentType(fds))
currentType(fds)
currentType(fds) <- value
fitMetrics(fds)
fitMetrics(fds) <- value
pseudocount(value = NULL)
hyperParams(fds, type = currentType(fds), all = FALSE)
bestQ(fds, type = currentType(fds))
dontWriteHDF5(fds)
dontWriteHDF5(fds) <- value
verbose(fds)
verbose(fds) <- value
```

**Arguments**

- **fds**
  - An FraserDataSet object.
- **type**
  - The type of psi (psi5, psi3 or theta)
- **value**
  - The new value to be assigned.
- **byGroup**
  - If TRUE, aggregation by donor/acceptor site will be done.
- **...**
  - Internally used parameters.
- **level**
  - Indicates if the retrieved p values should be adjusted on the donor/acceptor site-level (default) or if unadjusted junction-level p values should be returned.
- **filters**
  - A named list giving the filters that were applied for masking during p value correction. Used for storing and retrieving the correct set of requested p values.
- **dist**
  - Distribution for which the p-values should be extracted.
- **subsetName**
  - The name of a subset of genes of interest for which FDR corrected pvalues were previously computed. Default is NULL (using transcriptome-wide FDR corrected pvalues).
getter_setter_functions

**all**  Logical value indicating whether \texttt{hyperParams(fds)} should return the results of all evaluated parameter combinations or only for the optimal parameter combination.

**Value**
A (delayed) matrix or vector dependent on the type of data retrieved.

**Functions**

- **featureExclusionMask()**: Retrieves a logical vector indicating for each junction whether it is included or excluded during the fitting procedure.
- **featureExclusionMask(fds, type = currentType(fds)) <- value**: To remove certain junctions from being used in the train step of the encoding dimension we can set the \texttt{featureExclusionMask} vector to \texttt{FALSE}. This can be helpfull if we have local linkage between features which we do not want to model by the autoencoder.
- **rho()**: Returns the fitted rho values for the beta-binomial distribution
- **zScores()**: This returns the calculated z-scores.
- **pVals()**: This returns the calculated p-values.
- **padjVals()**: This returns the adjusted p-values.
- **availableFDRsubsets()**: This returns the names of FDR subsets for which adjusted p values have been calculated.
- **predictedMeans()**: This returns the fitted mu (i.e. psi) values.
- **deltaPsiValue()**: Returns the difference between the observed and the fitted psi values.
- **currentType()**: Returns the psi type that is used within several methods in the FRASER package (defaults to jaccard).
- **currentType(fds) <- value**: Sets the psi type that is to be used within several methods in the FRASER package.
- **fitMetrics()**: Returns the splice metrics that will be fitted (defaults to jaccard, used within several methods in the FRASER package).
- **fitMetrics(fds) <- value**: Sets the splice metrics that will be fitted (used within several methods in the FRASER package).
- **pseudocount()**: Sets and returns the pseudo count used within the FRASER fitting procedure.
- **hyperParams()**: This returns the results of the hyperparameter optimization NULL if the hyperparameter optimization was not run yet.
- **bestQ()**: This returns the optimal size of the latent space according to the hyperparameter optimization or a simple estimate of about a tenth of the number of samples if the hyperparameter optimization was not run yet.
- **dontWriteHDF5()**: Gets the current value of whether the assays should be stored as hdf5 files.
- **dontWriteHDF5(fds) <- value**: Sets whether the assays should be stored as hdf5 files.
- **verbose()**: Dependent on the level of verbosity the algorithm reports more or less to the user. 0 means being quiet and 10 means everything.
- **verbose(fds) <- value**: Sets the verbosity level to a value between 0 and 10. 0 means being quiet and 10 means reporting everything.
Examples

fds <- createTestFraserDataSet()

# should assays be saved as hdf5?
dontWriteHDF5(fds)
dontWriteHDF5 <- TRUE

# get/set the splice metric for which results should be retrieved
currentType(fds) <- "jaccard"
currentType(fds)

# get fitted parameters
bestQ(fds)
predictedMeans(fds)
rho(fds)

# get statistics
pVals(fds)
padjVals(fds)

# zscore not calculated by default
fds <- calculateZscore(fds, type="jaccard")
zScores(fds)

# set and get pseudocount
pseudocount(4L)
pseudocount()

# retrieve or set a mask to exclude certain junctions in the fitting step
featureExclusionMask(fds, type="jaccard") <- sample(c(FALSE, TRUE), nrow(mcols(fds, type="jaccard")), replace=TRUE)
featureExclusionMask(fds, type="jaccard")

# controlling the verbosity level of the output of some algorithms
verbose(fds) <- 2
verbose(fds)

infectOutliers

Inject artificial outliers in an existing fds

Description

Inject artificial outliers in an existing fds

Usage

infectOutliers(
  fds,
  type = psiTypes,
freq = 0.001,
minDpsi = 0.2,
minCoverage = 2,
deltaDistr = "uniformDistr",
verbose = FALSE,
method = c("samplePSI", "meanPSI", "simulatedPSI"),
BPPARAM = bpparam()
)

Arguments

fds FraserDataSet
type The psi type
freq The injection frequency.
minDpsi The minimal delta psi with which outliers will be injected.
minCoverage The minimal total coverage (i.e. N) required for a junction to be considered for injection of an outlier.
deltaDistr The distribution from which the delta psi value of the injections is drawn (default: uniform distribution).
verbose Should additional information be printed during computation?
method Defines by which method the new psi of injections is computed, i.e. to which value the delta psi of the injection is added: "meanPSI" for adding to the mean psi of the junction over all samples or "samplePSI" to add to the psi value of the junction in the specific sample. "simulatedPSI" is only possible if a simulated dataset is used.
BPPARAM A BiocParallel object to run the computation in parallel

Value

FraserDataSet

Examples

# A generic dataset
fds <- makeSimulatedFraserDataSet()
fds <- calculatePSIValues(fds)
fds <- injectOutliers(fds, minDpsi=0.2, freq=1E-3)

K Getter/setter for count data

Description

Getter/setter for count data
setter for count data
Usage

K(fds, type = currentType(fds))
N(fds, type = currentType(fds))

## S4 method for signature 'FraserDataSet'
counts(object, type = currentType(object), side = c("ofInterest", "otherSide"))

## S4 replacement method for signature 'FraserDataSet,ANY'
counts(
  object,
  type = currentType(object),
  side = c("ofInterest", "otherSide"),
)
<- value

Arguments

fds, object FraserDataSet

Type

The psi type.

Side

"ofInterest" for junction counts, "other" for sum of counts of all other junctions at the same donor site (psi5) or acceptor site (psi3), respectively.

Further parameters that are passed to assays(object, ...)

Value

FraserDataSet

Examples

fds <- createTestFraserDataSet()

counts(fds, side="ofInterest")
counts(fds, type="jaccard", side="other")
head(K(fds))
head(K(fds, type="psi5"))
head(K(fds, type="psi3"))
head(N(fds, type="theta"))
Description

retrieve the length of the object (aka number of junctions)

Usage

```r
## S4 method for signature 'FraserDataSet'
length(x)
```

Arguments

- `x`: FraserDataSet

Value

Length of the object.

loadFraserDataSet

Loading/Saving FraserDataSets

Description

This is a convenient function to load and save a FraserDataSet object. It looks and saves the FraserDataSet objects and HDF5 files on disk under the given working dir. Internally it uses HDF5 files for all assays.

Usage

```r
loadFraserDataSet(dir, name = NULL, file = NULL, upgrade = FALSE)
```

```r
saveFraserDataSet(fds, dir = NULL, name = NULL, rewrite = FALSE)
```

Arguments

- `dir`: A path where to save the objects (replaces the working directory)
- `name`: The analysis name of the project (saved within the 'dir')
- `file`: The file path to the fds-object.RDS file that should be loaded.
- `upgrade`: Should the version of the loaded object be updated?
- `fds`: A FraserDataSet object ot be saved
- `rewrite`: logical if the object should be rewritten. This makes sense if you have filtered or subsetted the object and want to save only the subsetted version
makeSimulatedFraserDataSet

Create an simulated example data set for FRASER

Description

Simulates a data set based on random counts following a beta binomial (or Dirichlet-Multinomial) distribution.

Usage

makeSimulatedFraserDataSet(
  m = 100,
  j = 500,
  q = 10,
  distribution = c("BB", "DM"),
  ...
)

Arguments

m       Number of simulated samples
j       Number of simulated junctions
q       number of simulated latent variables.
distribution Either "BB" for a beta-binomial simulation or "DM" for a dirichlet-multinomial simulation.
...     Further arguments used to construct the FraserDataSet.
Value

An FraserDataSet containing an example dataset based on simulated data

Examples

# A generic dataset
fds1 <- makeSimulatedFraserDataSet()
fds1

# A generic dataset with specified sample size and injection method
fds2 <- makeSimulatedFraserDataSet(m=10, j=100, q=3)
fds2

mergeExternalData Merge external data

Description

To boost its own sequencing data, one can download existing and precounted data. This function merges the existing FraserDataSet with external count data.

Usage

mergeExternalData(fds, countFiles, sampleIDs, annotation = NULL)

Arguments

fds A FraserDataSet
countFiles A character vector of file names pointing to the external count data. The vector has to be names or the files have to start with k_j, k_theta, n_psi3, n_psi5, n_theta.
sampleIDs The samples to be merged from the external data.
anotation A sample annotation of the external data (optional).

Details

For more details on existing datasets have a look at: <https://github.com/gagneurlab/drop#datasets>
Since FRASER can not hand NA values, the merge will return only the intersecting regions and will drop any non overlapping features. This has to be kept in mind when analysing rare disease samples.

Value

Merged FraserDataSet object.
Examples
ann <- fread(system.file("extdata", "externalCounts", "annotation.tsv.gz", package="FRASER"))
ctsFiles <- list.files(full.names = TRUE, pattern="counts", system.file("extdata", "externalCounts", package="FRASER"))
fds <- createTestFraserDataSet()
fds_merged <- mergeExternalData(fds, ctsFiles, anno[,sampleID], anno)

K(fds, "psi5")
K(fds_merged, "psi5")

optimHyperParams

Find optimal encoding dimension

Description
Finds the optimal encoding dimension by injecting artificial splicing outlier ratios while maximizing the precision-recall curve.

Usage
optimHyperParams(
  fds,
  type = psiTypes,
  implementation = "PCA",
  q_param = getEncDimRange(fds),
  noise_param = 0,
  minDeltaPsi = 0.1,
  iterations = 5,
  setSubset = 50000,
  injectFreq = 0.01,
  BPPARAM = bpparam(),
  internalThreads = 1,
  plot = TRUE,
  delayed = ifelse(ncol(fds) <= 300, FALSE, TRUE),
...
)

Arguments
fds A FraserDataSet object
type The type of PSI (jaccard, psi5, psi3 or theta for theta/splicing efficiency)
implementation The method that should be used to correct for confounders.
q_param Vector specifying which values of q should be tested
optimHyperParams

- **noise_param**: Vector specifying which noise levels should be tested.
- **minDeltaPsi**: Minimal delta psi of an intron to be considered a variable intron.
- **iterations**: The maximal number of iterations. When the autoencoder has not yet converged after these number of iterations, the fit stops anyway.
- **setSubset**: The size of the subset of the most variable introns that should be used for the hyperparameter optimization.
- **injectFreq**: The frequency with which outliers are injected into the data.
- **BPPARAM**: The BiocParallel parameters for the parallelization
- **internalThreads**: The number of threads used internally.
- **plot**: If TRUE, a plot of the area under the curve and the model loss for each evaluated parameter combination will be displayed after the hyperparameter optimization finishes.
- **delayed**: If FALSE, count matrices will be loaded into memory (faster calculations), otherwise the function works on the delayedMatrix representations (more memory efficient). The default value depends on the number of samples in the fds-object.
- **...**: Additional parameters passed to injectOutliers.

**Value**

FraserDataSet

**See Also**

FRASER

**Examples**

```r
# generate data
fds <- makeSimulatedFraserDataSet(m=15, j=20)
fds <- calculatePSIValues(fds)

# run hyperparameter optimization
fds <- optimHyperParams(fds, type="jaccard", q_param=c(2, 5))

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

Visualization functions for FRASER

Description

The FRASER package provides multiple functions to visualize the data and the results of a full data set analysis.

Plots the p values over the delta psi values, known as volcano plot. Visualizes per sample the outliers. By type and aggregate by gene if requested.

Plot the number of aberrant events per samples

Plots the observed split reads of the junction of interest over all reads coming from the given donor/acceptor.

Plots the observed values of the splice metric across samples for a junction of interest.

Plots the expected psi value over the observed psi value of the given junction.

Plots the quantile-quantile plot

Histogram of the geometric mean per junction based on the filter status

Histogram of minimal delta psi per junction

Count correlation heatmap function

Usage

plotManhattan(object, ...)

## S4 method for signature 'FraserDataSet'
plotVolcano(
  object,
  sampleID,
  type = fitMetrics(object),
  basePlot = TRUE,
  aggregate = FALSE,
  main = NULL,
  label = NULL,
  deltaPsiCutoff = 0.1,
  padjCutoff = 0.1,
  subsetName = NULL,
  ...
)

## S4 method for signature 'FraserDataSet'
plotAberrantPerSample(
  object,
  main,
  type = fitMetrics(object),
  padjCutoff = 0.1,
deltaPsiCutoff = 0.1,
aggregate = TRUE,
subsetName = NULL,
BPPARAM = bpparam(),
...)

plotExpression(
fds,
type = fitMetrics(fds),
idx = NULL,
result = NULL,
colGroup = NULL,
basePlot = TRUE,
main = NULL,
label = "aberrant",
subsetName = NULL,
...)

plotSpliceMetricRank(
fds,
type = fitMetrics(fds),
idx = NULL,
result = NULL,
colGroup = NULL,
basePlot = TRUE,
main = NULL,
label = "aberrant",
subsetName = NULL,
...)

plotExpectedVsObservedPsi(
fds,
type = fitMetrics(fds),
idx = NULL,
result = NULL,
colGroup = NULL,
main = NULL,
basePlot = TRUE,
label = "aberrant",
subsetName = NULL,
...)

## S4 method for signature 'FraserDataSet'
plotQQ(
object,
type = NULL,
idx = NULL,
result = NULL,
aggregate = FALSE,
global = FALSE,
main = NULL,
conf.alpha = 0.05,
samplingPrecision = 3,
basePlot = TRUE,
label = "aberrant",
Ncpus = min(3, getDTthreads()),
subsetName = NULL,
"
## S4 method for signature 'FraserDataSet'
plotEncDimSearch(object, type = psiTypes, plotType = c("auc", "loss"))

plotFilterExpression(
  fds,
  bins = 200,
  legend.position = c(0.8, 0.8),
  onlyVariableIntrons = FALSE
)

plotFilterVariability(
  fds,
  bins = 200,
  legend.position = c(0.8, 0.8),
  onlyExpressedIntrons = FALSE
)

## S4 method for signature 'FraserDataSet'
plotCountCorHeatmap(
  object,
type = psiTypes,
logit = FALSE,
topN = 50000,
topJ = 5000,
minMedian = 1,
minCount = 10,
main = NULL,
normalized = FALSE,
show_rownames = FALSE,
show_colnames = FALSE,
minDeltaPsi = 0.1,
annotation_col = NA,
plotManhattan

annotation_row = NA,
border_color = NA,
nClust = 5,
plotType = c("sampleCorrelation", "junctionSample"),
sampleClustering = NULL,
plotMeanPsi = TRUE,
plotCov = TRUE,
...
)

plotBamCoverage(
    fds,
    gr,
    sampleID,
    control_samples = sample(samples(fds[, which(samples(fds) != sampleID)]), min(3, ncol(fds) - length(sampleID))),
    txdb = NULL,
    min_junction_count = 20,
    highlight_range = NULL,
    highlight_range_color = "firebrick",
    color_annotated = "gray",
    color_novel = "goldenrod3",
    color_sample_interest = "firebrick",
    color_control_samples = "dodgerblue4",
    toscale = c("exon", "gene", "none"),
    mar = c(2, 10, 0.1, 5),
    curvature_splicegraph = 1,
    curvature_coverage = 1,
    cex = 1,
    splicegraph_labels = c("genomic_range", "id", "name", "none"),
    splicegraph_position = c("top", "bottom"),
    ...
)

plotBamCoverageFromResultTable(
    fds,
    result,
    show_full_gene = FALSE,
    txdb = NULL,
    orgDb = NULL,
    res_gene_col = "hgncSymbol",
    res_geneid_type = "SYMBOL",
    txdb_geneid_type = "ENTREZID",
    left_extension = 1000,
    right_extension = 1000,
    ...
)
## S4 method for signature 'FraserDataSet'
plotManhattan(
  object,
  sampleID,
  value = "pvalue",
  type = fitMetrics(object),
  chr = NULL,
  main = paste0("sample: ", sampleID),
  chrColor = c("black", "darkgrey"),
  subsetName = NULL,
  ...
)

### Arguments

- **object, fds**: An *FraserDataSet* object.
- **...**: Additional parameters passed to plot() or plot_ly() if not stated otherwise in the details for each plot function.
- **sampleID**: A sample ID which should be plotted. Can also be a vector. Integers are treated as indices.
- **type**: The psi type: either psi5, psi3 or theta (for SE).
- **basePlot**: if TRUE (default), use the R base plot version, else use the plotly framework.
- **aggregate**: If TRUE, the pvalues are aggregated by gene (default), otherwise junction level pvalues are used (default for Q-Q plot).
- **main**: Title for the plot, if missing a default title will be used.
- **label**: Indicates the genes or samples that will be labelled in the plot (only for basePlot=TRUE). Setting label="aberrant" will label all aberrant genes or samples. Labelling can be turned off by setting label=NULL. The user can also provide a custom list of gene symbols or sample IDs.
- **padjCutoff, deltaPsiCutoff**: Significance or delta psi cutoff to mark outliers.
- **subsetName**: The name of a subset of genes of interest for which FDR corrected pvalues were previously computed. Those FDR values on the subset will then be used to determine aberrant status. Default is NULL (using transcriptome-wide FDR corrected pvalues).
- **BPPARAM**: BiocParallel parameter to use.
- **idx**: A junction site ID or gene ID or one of both, which should be plotted. Can also be a vector. Integers are treated as indices.
- **result**: The result table to be used by the method.
- **colGroup**: Group of samples that should be colored.
- **global**: Flag to plot a global Q-Q plot, default FALSE.
- **conf.alpha**: If set, a confidence interval is plotted, defaults to 0.05.
- **samplingPrecision**: Plot only non overlapping points in Q-Q plot to reduce number of points to plot. Defines the digits to round to.
Ncpus Number of cores to use.

plotType The type of plot that should be shown as character string. For plotEncDim-
Search, it has to be either "auc" for a plot of the area under the curve (AUC)
or "loss" for the model loss. For the correlation heatmap, it can be either
"sampleCorrelation" for a sample-sample correlation heatmap or "junctionSample"
for a junction-sample correlation heatmap.

bins Set the number of bins to be used in the histogram.
legend.position Set legend position (x and y coordinate), defaults to the top right corner.

onlyVariableIntrons Logical value indicating whether to show only introns that also pass the vari-
ability filter. Defaults to FALSE.

onlyExpressedIntrons Logical value indicating whether to show only introns that also pass the expres-
sion filter. Defaults to FALSE.

logit If TRUE, the default, psi values are plotted in logit space.
topN Top x most variable junctions that should be used for the calculation of sample
x sample correlations.
topJ Top x most variable junctions that should be displayed in the junction-sample
correlation heatmap. Only applies if plotType is "junctionSample".

minMedian, minCount, minDeltaPsi Minimal median \( m \geq 1 \), delta psi \( |\Delta \psi| > 0.1 \), read count \( n \geq 10 \) value of
a junction to be considered for the correlation heatmap.

normalized If TRUE, the normalized psi values are used, the default, otherwise the raw psi
calculate values

show_rownames, show_colnames Logical value indicating whether to show row or column names on the heatmap
axes.

annotation_col, annotation_row Row or column annotations that should be plotted on the heatmap.

border_color Sets the border color of the heatmap

nClust Number of clusters to show in the row and column dendrograms.
sampleClustering A clustering of the samples that should be used as an annotation of the heatmap.

plotMeanPsi, plotCov If TRUE, then the heatmap is annotated with the mean psi values or the junction
coverage.

gr A GRanges object indicating the genomic range that should be shown in plotBamCoverage.
control_samples The sampleIDs of the samples used as control in plotBamCoverage.
txdb A TxDb object giving the gene/transcript annotation to use.

min_junction_count The minimal junction count across samples required for a junction to appear in
the splicegraph and coverage tracks of plotBamCoverage.
highlight_range
A GenomicRanges or GenomicRangesList object of ranges to be highlighted in the splicegraph of plotBamCoverage.

highlight_range_color
The color of highlighted ranges in the splicegraph of plotBamCoverage.

color_annotated
The color for exons and junctions present in the given annotation (in the splicegraph of plotBamCoverage).

color_novel
The color for novel exons and junctions not present in the given annotation (in the splicegraph of plotBamCoverage).

color_sample_interest
The color in plotBamCoverage for the sample of interest.

color_control_samples
The color in plotBamCoverage for the samples used as controls.

toscale
In plotBamCoverage, indicates which part of the plotted region should be drawn to scale. Possible values are 'exon' (exonic regions are drawn to scale), 'gene' (both exonic and intronic regions are drawn to scale) or 'none' (exonic and intronic regions have constant length) (see SGSeq package).

mar
The margin of the plot area for plotBamCoverage (b,l,t,r).

curvature_splicegraph
The curvature of the junction arcs in the splicegraph in plotBamCoverage. Decrease this value for flatter arcs and increase it for steeper arcs.

curvature_coverage
The curvature of the junction arcs in the coverage tracks of plotBamCoverage. Decrease this value for flatter arcs and increase it for steeper arcs.

cex
For controlling the size of text and numbers in plotBamCoverage.

splicegraph_labels
Indicated the format of exon/splice junction labels in the splicegraph of plotBamCoverage. Possible values are 'genomic_range' (gives the start position of the first exon and the end position of the last exon that are shown), 'id' (format E1,... J1,...), 'name' (format type:chromosome:start-end:strand for each feature), 'none' for no labels (see SGSeq package).

splicegraph_position
The position of the splicegraph relative to the coverage tracks in plotBamCoverage. Possible values are 'top' (default) and 'bottom'.

show_full_gene
Should the full genomic range of the gene be shown in plotBamCoverageFromResultTable (default: FALSE)? If FALSE, only a certain region (see parameters left_extension and right_extension) around the outlier junction is shown.

orgDb
A OrgDb object giving the mapping of gene ids and symbols.

res_gene_col
The column name in the given results table that contains the gene annotation.

res_geneid_type
The type of gene annotation in the results table in res_gene_col (e.g. SYMBOL or ENTREZID etc.). This information is needed for mapping between the results table and the provided annotation in the txdb object.
txdb_geneid_type
The type of gene_id present in genes(txdb) (e.g. ENTREZID). This information is needed for mapping between the results table and the provided annotation in the txdb object.

left_extension
Indicating how far the plotted range around the outlier junction should be extended to the left in plotBamCoverageFromResultTable.

right_extension
Indicating how far the plotted range around the outlier junction should be extended to the right in plotBamCoverageFromResultTable.

value
Indicates which assay is shown in the manhattan plot. Defaults to 'pvalue'. Other options are 'deltaPsi' and 'zScore'.

chr
Vector of chromosome names to show in plotManhattan. The default is to show all chromosomes.

chrColor
Interchanging colors by chromosome for plotManhattan.

Details
This is the list of all plotting function provided by FRASER:

- plotAberrantPerSample()
- plotVolcano()
- plotExpression()
- plotQQ()
- plotExpectedVsObservedPsi()
- plotCountCorHeatmap()
- plotFilterExpression()
- plotFilterVariability()
- plotEncDimSearch()
- plotBamCoverage()
- plotBamCoverageFromResultTable()
- plotManhattan()
- plotSpliceMetricRank()

For a detailed description of each plot function please see the details. Most of the functions share the same parameters.

plotAberrantPerSample: The number of aberrant events per sample are plotted sorted by rank. The ... parameters are passed on to the aberrant function.

plotVolcano: the volcano plot is sample-centric. It plots for a given sample and psi type the negative log10 nominal P-values against the delta psi values for all splice sites or aggregates by gene if requested.

plotExpression: This function plots for a given site the read count at this site (i.e. K) against the total coverage (i.e. N) for the given psi type ($\psi_5, \psi_3, or \theta$ (SE)) for all samples.

plotQQ: the quantile-quantile plot for a given gene or if global is set to TRUE over the full data set. Here the observed P-values are plotted against the expected ones in the negative log10 space.
plotExpectedVsObservedPsi: A scatter plot of the observed psi against the predicted psi for a given site.

plotSpliceMetricRank: This function plots for a given intron the observed values of the selected splice metric against the sample rank.

plotCountCorHeatmap: The correlation heatmap of the count data either of the full data set (i.e. sample-sample correlations) or of the top x most variable junctions (i.e. junction-sample correlations). By default the values are log transformed and row centered. The ... arguments are passed to the pheatmap function.

plotFilterExpression: The distribution of FPKM values. If the FraserDataSet object contains the passedFilter column, it will plot both FPKM distributions for the expressed introns and for the filtered introns.

plotFilterVariability: The distribution of maximal delta Psi values. If the FraserDataSet object contains the passedFilter column, it will plot both maximal delta Psi distributions for the variable introns and for the filtered (i.e. non-variable) introns.

plotEncDimSearch: Visualization of the hyperparameter optimization. It plots the encoding dimension against the achieved loss (area under the precision-recall curve). From this plot the optimum should be choosen for the q in fitting process.

plotManhattan: A Manhattan plot showing the junction pvalues by genomic position. Useful to identify if outliers cluster by genomic position.

plotBamCoverage: A sashimi plot showing the read coverage from the underlying bam files for a given genomic range and sampleIDs.

plotBamCoverageFromResultTable: A sashimi plot showing the read coverage from the underlying bam files for a row in the results table. Can either show the full range of the gene with the outlier junction or only a certain region around the outlier.

Value

If base R graphics are used nothing is returned else the plotly or the gplot object is returned.

Examples

```r
# create full FRASER object
fds <- makeSimulatedFraserDataSet(m=40, j=200)
fds <- calculatePSIValues(fds)
fds <- filterExpressionAndVariability(fds, filter=FALSE)
# this step should be done for more dimensions in practice
fds <- optimHyperParams(fds, "jaccard", q_param=c(2,5,10,25))

# assign gene names to show functionality on test dataset
# use fds <- annotateRanges(fds) on real data
mcols(fds, type="j")$hgnc_symbol <-
  paste0("gene", sample(1:25, nrow(fds), replace=TRUE))

# fit and calculate pvalues
genesOfInterest <- rep(list(paste0("gene", sample(1:25, 10))), 4)
names(genesOfInterest) <- c("sample1", "sample6", "sample15", "sample23")
fds <- FRASER(fds, subets=list("testSet"=genesOfInterest))
```
# QC plotting
plotFilterExpression(fds)
plotFilterVariability(fds)
plotCountCorHeatmap(fds, "jaccard")
plotCountCorHeatmap(fds, "jaccard", normalized=TRUE)
plotEncDimSearch(fds, type="jaccard")

# extract results
plotAberrantPerSample(fds, aggregate=FALSE)
plotAberrantPerSample(fds, aggregate=TRUE, subsetName="testSet")
plotVolcano(fds, "sample2", "jaccard", label="aberrant")
plotVolcano(fds, "sample1", "jaccard", aggregate=TRUE, subsetName="testSet")

# dive into gene/sample level results
res <- as.data.table(results(fds))
res
plotExpression(fds, result=res[1])
plotQQ(fds, result=res[1])
plotExpectedVsObservedPsi(fds, res=res[1])
plotSpliceMetricRank(fds, res=res[1])

# other ways to call these plotting functions
plotExpression(fds, idx=10, sampleID="sample1", type="jaccard")
plotExpression(fds, result=res[1], subsetName="testSet")
plotQQ(fds, idx=10, sampleID="sample1", type="jaccard")
plotQQ(fds, result=res[1], subsetName="testSet")
plotExpectedVsObservedPsi(fds, idx=10, sampleID="sample1", type="jaccard")
plotExpectedVsObservedPsi(fds, result=res[1], subsetName="testSet")
plotSpliceMetricRank(fds, idx=10, sampleID="sample1", type="jaccard")
plotSpliceMetricRank(fds, result=res[1], subsetName="testSet")

# create manhattan plot of pvalues by genomic position
if(require(ggbio)){
  plotManhattan(fds, type="jaccard", sampleID="sample10")
}

# plot splice graph and coverage from bam files in a given region
if(require(SGSeq)){
  fds <- createTestFraserSettings()
  gr <- GRanges(seqnames="chr19",
                IRanges(start=7587496, end=7598895),
                strand="+")
  plotBamCoverage(fds, gr=gr, 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()
  require(TxDb.Hsapiens.UCSC.hg19.knownGene)
  txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
  require(org.Hs.eg.db)
  orgDb <- org.Hs.eg.db
res <- results(fds, padjCutoff=NA, deltaPsiCutoff=NA)
res_dt <- as.data.table(res)
res_dt <- res_dt[sampleID == "sample2",]

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

# plot only certain range around outlier junction
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")

---

potentialImpactAnnotations

**Additional result annotations**

**Description**

These functions work on the result table and add additional annotations to the reported introns: the type of potential impact on splicing (e.g. exon skipping, exon truncation, ...), potential occurrence of frameshift, overlap with UTR regions as well as a flag for introns that are located in blacklist regions of the genome.

*annotateIntronReferenceOverlap* adds basic annotations to the fds for each intron based on the overlap of the intron’s location with the reference annotation. Has to be run before the result table is created so that the new column can be included in it (see examples).

*annotatePotentialImpact* annotates each intron in the results table with the type of potential impact on splicing and potential occurrence of frameshift (likely, unlikely, inconclusive). Can also calculate overlap with annotated UTR regions. Potential impact can be: annotatedIntron_increasedUsage, annotatedIntron_reducedUsage, exonTruncation, exonElongation, exonTruncation&Elongation, exonSkipping, splicingBeyondGene, multigenicSplicing, downstreamOfNearestGene, upstreamOfNearestGene, complex (everything else). Splice sites (theta metric) annotations indicate how the splice site is located with respect to the reference annotation. The annotated types are: annotated-SpliceSite, exonicRegion, intronicRegion.

*flagBlacklistRegions* flags introns in the results table on whether or not they are located in a blacklist region of the genome. By default, the blacklist regions as reported in *Amemiya, Kundaje & Boyle* (2019) and downloaded from here are used.

**Usage**

annotateIntronReferenceOverlap(fds, txdb, BPPARAM = bpparam())

annotatePotentialImpact(
result,
  txdb,
  fds,
  addPotentialImpact = TRUE,
  addUTRoverlap = TRUE,
  minoverlap = 5,
  BPPARAM = bpparam()
)

flagBlacklistRegions(
  result,
  blacklist_regions = NULL,
  assemblyVersion = c("hg19", "hg38"),
  minoverlap = 5
)

Arguments

- **fds**: A FraserDataSet
- **txdb**: A txdb object providing the reference annotation.
- **BPPARAM**: For controlling parallelization behavior. Defaults to bpparam().
- **result**: A result table as generated by FRASER, including the column annotatedJunction as generated by the function annotateIntronReferenceOverlap.
- **addPotentialImpact**: Logical, indicating if the type of the potential impact should be added to the results table. Defaults to TRUE.
- **addUTRoverlap**: Logical, indicating if the overlap with UTR regions should checked and added to the results table. Defaults to TRUE.
- **minoverlap**: Integer value defining the number of base pairs around the splice site that need to overlap with UTR or blacklist region, respectively, to be considered matching. Defaults to 5 bp.
- **blacklist_regions**: A BED file that contains the blacklist regions. If NULL (default), the BED files that are packaged with FRASER are used (see Details for more information).
- **assemblyVersion**: Indicates the genome assembly version of the intron coordinates. Only used if blacklist_regions is NULL. For other versions, please provide the BED file containing the blacklist regions directly.

Value

An annotated FraserDataSet or results table, respectively

Functions

- **annotateIntronReferenceOverlap()**: This method calculates basic annotations based on overlap with the reference annotation (start, end, none, both) for the full fds. The overlap type is added as a new column annotatedJunction in mcols(fds).
annotatePotentialImpact(): This method annotates the splice event type to junctions in the given results table.

flagBlacklistRegions(): This method flags all introns and splice sites in the given results table for which at least one splice site (donor or acceptor) is located in a blacklist region. Blacklist regions of the genome are determined from the provided BED file.

Examples

```r
# get data, fit and compute p-values and z-scores
fds <- createTestFraserDataSet()

# load reference annotation
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

# add basic annotations for overlap with the reference annotation
# run this function before creating the results table
fds <- annotateIntronReferenceOverlap(fds, txdb)

# extract results: for this small example dataset, no cutoffs used
# to get some results
res <- results(fds, padjCutoff=NA, deltaPsiCutoff=NA)

# annotate the type of potential impact on splicing and UTR overlap
res <- annotatePotentialImpact(result=res, txdb=txdb, fds=fds)

# annotate overlap with blacklist regions
res <- flagBlacklistRegions(result=res, assemblyVersion="hg19")

# show results table containing additional annotations
res
```

<table>
<thead>
<tr>
<th>psiTypes</th>
<th>Available splice metrics</th>
</tr>
</thead>
</table>

Description

Available splice metrics

Usage

psiTypes

Format

An object of class character of length 4.
Examples

  # to show all available splice metrics:
  psiTypes

Description

The result function extracts the results from the given analysis object based on the given options and cutoffs. The aberrant function extracts aberrant splicing events based on the given cutoffs.

Usage

```r
## S4 method for signature 'FraserDataSet'
results(
  object,
  sampleIDs = samples(object),
  padjCutoff = 0.1,
  deltaPsiCutoff = 0.1,
  rhoCutoff = NA,
  aggregate = FALSE,
  collapse = FALSE,
  minCount = 5,
  psiType = psiTypes,
  geneColumn = "hgnc_symbol",
  all = FALSE,
  returnTranscriptomewideResults = TRUE,
  additionalColumns = NULL,
  BPPARAM = bpparam()
)

## S4 method for signature 'FraserDataSet'
aberrant(
  object,
  type = fitMetrics(object),
  padjCutoff = 0.1,
  deltaPsiCutoff = 0.1,
  minCount = 5,
  rhoCutoff = NA,
  by = c("none", "sample", "feature"),
  aggregate = FALSE,
  geneColumn = "hgnc_symbol",
  subsetName = NULL,
  all = FALSE,
)```
Arguments

- **object**
  A *FraserDataSet* object

- **sampleIDs**
  A vector of sample IDs for which results should be retrieved

- **padjCutoff**
  The FDR cutoff to be applied or NA if not requested.

- **deltaPsiCutoff**
  The cutoff on delta psi or NA if not requested.

- **rhoCutoff**
  The cutoff value on the fitted rho value (overdispersion parameter of the beta-binomial) above which junctions are filtered

- **aggregate**
  If TRUE the returned object is aggregated to the feature level (i.e. gene level).

- **collapse**
  Only takes effect if aggregate=TRUE. If TRUE, collapses results across the different psi types to return only one row per feature (gene) and sample.

- **minCount**
  The minimum count value of the total coverage of an intron to be considered as significant.

- **psiType**
  The psi types for which the results should be retrieved.

- **geneColumn**
  The column name of the column that has the gene annotation that will be used for gene-level pvalue computation.

- **all**
  By default FALSE, only significant introns (or genes) are listed in the results. If TRUE, results are assembled for all samples and introns/genes regardless of significance.

- **returnTranscriptomewideResults**
  If FDR corrected pvalues for subsets of genes of interest have been calculated, this parameter indicates whether additionally the transcriptome-wide results should be returned as well (default), or whether only results for those subsets should be retrieved.

- **additionalColumns**
  Character vector containing the names of additional columns from mcols(fds) that should appear in the result table (e.g. ensembl_gene_id). Default is NULL, so no additional columns are included.

- **BPPARAM**
  The BiocParallel parameter.

- **type**
  Splicing type (psi5, psi3 or theta)

- **by**
  By default none which means no grouping. But if sample or feature is specified the sum by sample or feature is returned

- **subsetName**
  The name of a subset of genes of interest for which FDR corrected pvalues were previously computed. Those FDR values on the subset will then be used to determine aberrant status. Default is NULL (using transcriptome-wide FDR corrected pvalues).

... Further arguments can be passed to the method. If "n", "padjVals", "dPsi" or "rhoVals" are given, the values of those arguments are used to define the aberrant events.
Value

For results: GRanges object containing significant results. For aberrant: Either a of logical values of size introns/genes x samples if "by" is NA or a vector with the number of aberrant events per sample or feature depending on the value of "by"

Examples

# get data, fit and compute p-values and z-scores
fds <- createTestFraserDataSet()

# extract results: for this example dataset, no cutoffs are used to
# show the output of the results function
res <- results(fds, all=TRUE)
res

# aggregate the results by genes (gene symbols need to be annotated first
# using annotateRanges() function)
results(fds, padjCutoff=NA, deltaPsiCutoff=0.1, aggregate=TRUE)

# aggregate the results by genes and collapse over all psi types to obtain
# only one row per gene in the results table
results(fds, padjCutoff=NA, deltaPsiCutoff=0.1, aggregate=TRUE,
collapse=TRUE)

# get aberrant events per sample: on the example data, nothing is aberrant
# based on the adjusted p-value
aberrant(fds, type="jaccard", by="sample")

# get aberrant events per gene (first annotate gene symbols)
fds <- annotateRangesWithTxDb(fds)
aberrant(fds, type="jaccard", by="feature", padjCutoff=NA, aggregate=TRUE)

# find aberrant junctions/splice sites
aberrant(fds, type="jaccard")

# retrieve results limiting FDR correction to only a subset of genes
# first, we need to create a list of genes per sample that will be tested
geneList <- list('sample1'=c("TIMMDC1"), 'sample2'=c("MCOLN1"))
fds <- calculatePadjValues(fds, type="jaccard",
subsets=list("exampleSubset"=geneList))
results(fds, all=TRUE, returnTranscriptomewideResults=FALSE)

samples

Getter/Setter methods for the FraserDataSet

Description

The following methods are getter and setter methods to extract or set certain values of a FraserDataSet object.
samples sets or gets the sample IDs; condition: nonSplicedReads return a RangedSummarizedExperiment object containing the counts for the non spliced reads overlapping splice sites in the fds.

Usage

```r
samples(object)
samples(object) <- value
condition(object)
condition(object) <- value
bamFile(object)
bamFile(object) <- value
name(object)
name(object) <- value
strandSpecific(object)
strandSpecific(object) <- value
pairedEnd(object)
pairedEnd(object) <- value
workingDir(object)
workingDir(object) <- value
scanBamParam(object)
scanBamParam(object) <- value
nonSplicedReads(object)
nonSplicedReads(object) <- value
```

## S4 method for signature 'FraserDataSet'
samples(object)

## S4 replacement method for signature 'FraserDataSet'
samples(object) <- value

## S4 method for signature 'FraserDataSet'
condition(object)

## S4 replacement method for signature 'FraserDataSet'
condition(object) <- value

## S4 method for signature 'FraserDataSet'
bamFile(object)

## S4 replacement method for signature 'FraserDataSet'
bamFile(object) <- value

## S4 method for signature 'FraserDataSet'
name(object)

## S4 replacement method for signature 'FraserDataSet'
name(object) <- value

## S4 method for signature 'FraserDataSet'
workingDir(object)

## S4 replacement method for signature 'FraserDataSet'
workingDir(object) <- value

## S4 method for signature 'FraserDataSet'
strandSpecific(object)

## S4 replacement method for signature 'FraserDataSet'
strandSpecific(object) <- value

## S4 method for signature 'FraserDataSet'
pairedEnd(object)

## S4 replacement method for signature 'FraserDataSet'
pairedEnd(object) <- value

## S4 method for signature 'FraserDataSet'
scanBamParam(object)

## S4 replacement method for signature 'FraserDataSet'
scanBamParam(object) <- value

## S4 method for signature 'FraserDataSet'
nonSplicedReads(object)

## S4 replacement method for signature 'FraserDataSet'
nonSplicedReads(object) <- value

FRASER.mcols.get(x, type = NULL, ...)

samples
FRASER.rowRanges.get(x, type = NULL, ...)

mapSeqlevels(fds, style = "UCSC", ...)

Arguments

object  A FraserDataSet object.
value   The new value that should replace the current one.
x       A FraserDataSet object.
type    The psi type (psi3, psi5 or theta)
...     Further parameters. For mapSeqLevels: further parameters passed to GenomeInfoDb::mapSeqlevels().

fds     FraserDataSet
style   The style of the chromosome names.

Value

Getter method return the respective current value.

Author(s)

Christian Mertes <mertes@in.tum.de>
Ines Scheller <scheller@in.tum.de>

Examples

def <- createTestFraserDataSet()
samples(def)
samples(def) <- 1:dim(def)[2]
condition(def)
condition(def) <- 1:dim(def)[2]
bamFile(def) # file.paths or objects of class BamFile
bamFile(def) <- file.path("bamfiles", samples(def), "rna-seq.bam")
name(def)
name(def) <- "My Analysis"
workingDir(def)
workingDir(def) <- tempdir()
strandSpecific(def)
strandSpecific(def) <- TRUE
strandSpecific(def) <- "reverse"
strandSpecific(def)
scanBamParam(def)
scanBamParam(def) <- ScanBamParam(mapqFilter=30)
nonSplicedReads(def)
rowRanges(def)
rowRanges(def, type="theta")
mcols(def, type="psi5")
mcols(def, type="theta")
Description

Providing subsetting by indices through the single-bracket operator

Usage

```r
## S3 method for class 'FRASER'
subset(x, i, j, by = c("j", "ss"), ..., drop = FALSE)

## S4 method for signature 'FraserDataSet,ANY,ANY,ANY'
x[i, j, by = c("j", "ss"), ..., drop = FALSE]
```

Arguments

- `x`: A FraserDataSet object
- `i`: A integer vector to subset the rows/ranges
- `j`: A integer vector to subset the columns/samples
- `by`: a character (j or ss) defining if we subset by junctions or splice sites
- `...`: Parameters currently not used or passed on
- `drop`: No dimension reduction is done. And the drop parameter is currently not used at all.

Value

A subsetted FraserDataSet object

Examples

```r
fds <- createTestFraserDataSet()
fds[1:10,2:3]
fds[,samples(fds) %in% c("sample1", "sample2")]
fds[1:10,by="ss"]
```
Index

* datasets
  psiTypes, 44
  [,FraserDataSet, ANY, ANY, ANY-method (subset.FRASER), 51
  ‘featureExclusionMask<-‘
    (getter_setter_functions), 21
aberrant, 39
aberrant, FraserDataSet-method
  (results, FraserDataSet-method), 45
addCountsToFraserDataSet, 10, 11
addCountsToFraserDataSet (countRNA), 7
annotateIntronReferenceOverlap, 42
annotateIntronReferenceOverlap
  (potentialImpactAnnotations), 42
annotatePotentialImpact, 42
annotatePotentialImpact
  (potentialImpactAnnotations), 42
annotateRanges, 3
annotateRangesWithTxDb
  (annotateRanges), 3
assayNames, FraserDataSet-method, 5
assays, FraserDataSet-method, 5
assays<-, FraserDataSet, DelayedMatrix-method
  (assays, FraserDataSet-method), 5
assays<-, FraserDataSet, list-method
  (assays, FraserDataSet-method), 5
assays<-, FraserDataSet, SimpleList-method
  (assays, FraserDataSet-method), 5
available.genomes, 9
availableFDRsubsets
  (getter_setter_functions), 21
bamFile (samples), 47
bamFile, FraserDataSet-method (samples), 47
bamFile<- (samples), 47
bamFile<-, FraserDataSet-method (samples), 47
bestQ (getter_setter_functions), 21
calculatePadjValues (FRASER), 16
calculatePadjValuesOnSubset (FRASER), 16
calculatePSIValues, 6
calculatePvalues (FRASER), 16
calculateZscore (FRASER), 16
condition (samples), 47
condition, FraserDataSet-method
  (samples), 47
condition<-(samples), 47
condition<-, FraserDataSet-method
  (samples), 47
countNonSplicedReads, 10, 11
countNonSplicedReads (countRNA), 7
countRNA, 7
countRNAData, 10
countRNAData (countRNA), 7
counts, FraserDataSet-method (K), 25
counts<-, FraserDataSet, ANY-method (K), 25
countSplitReads, 10, 11
countSplitReads (countRNA), 7
createTestFraserDataSet
  (createTestFraserSettings), 11
currentType (getter_setter_functions), 21
currentType<-
  (getter_setter_functions), 21
deltaPsiValue
  (getter_setter_functions), 21
dontWriteHDF5
  (getter_setter_functions), 21
don'tWriteHDF5<-
  (getter_setter_functions), 21
fds-methods (samples), 47
featureExclusionMask
  (getter_setter_functions), 21
featureExclusionMask,
  (getter_setter_functions), 21
featureExclusionMask<-
  (getter_setter_functions), 21
filterExpression, FraserDataSet-method
  (filterVariability), 12
filterExpressionAndVariability
  (filterVariability), 12
filtering (filterVariability), 12
filterVariability, FraserDataSet-method
  (filterVariability), 12
fit, 14, 18
fitMetrics (getter_setter_functions), 21
fitMetrics<-
  (getter_setter_functions), 21
flagBlacklistRegions, 42
flagBlacklistRegions
  (potentialImpactAnnotations), 42
FRASER, 14, 15, 16, 31
FRASER.mcols.get (samples), 47
FRASER.rowRanges.get (samples), 47
FraserDataSet, 6, 9, 10, 13, 15, 17, 19, 20, 30, 36, 46
FraserDataSet-class, 21
getBSgenome, 9
getNonSplitReadCountsForAllSamples, 10, 11
getNonSplitReadCountsForAllSamples
  (countRNA), 7
getSplitReadCountsForAllSamples, 10
getSplitReadCountsForAllSamples
  (countRNA), 7
ger_getter_setter_functions, 21
hyperParams (getter_setter_functions), 21
injectionOutliers, 24
keytypes, 4
length, FraserDataSet-method, 27
loadFraserDataSet, 27
makeSimulatedFraserDataSet, 28
mapSeqlevels (samples), 47
mergeCounts, 10, 11
mergeCounts (countRNA), 7
mergeExternalData, 29
N (K), 25
name (samples), 47
name, FraserDataSet-method (samples), 47
name<-, (samples), 47
name<-, FraserDataSet-method (samples), 47
nonSplicedReads (samples), 47
nonSplicedReads, FraserDataSet-method
  (samples), 47
nonSplicedReads<-, (samples), 47
nonSplicedReads<-, FraserDataSet-method
  (samples), 47
optimHyperParams, 15, 17, 30
padjVals (getter_setter_functions), 21
padjVals, (getter_setter_functions), 21
pairedEnd (samples), 47
pairedEnd, FraserDataSet-method
  (samples), 47
pairedEnd<-, (samples), 47
pairedEnd<-, FraserDataSet-method
  (samples), 47
pheatmap, 40
plotAberrantPerSample (plotManhattan), 32
plotAberrantPerSample, FraserDataSet-method
  (plotManhattan), 32
plotBamCoverage (plotManhattan), 32
plotBamCoverageFromResultTable
  (plotManhattan), 32
plotCountCorHeatmap, FraserDataSet-method
  (plotManhattan), 32
plotEncDimSearch (plotManhattan), 32
plotEncDimSearch, FraserDataSet-method
  (plotManhattan), 32
plotExpectedVsObservedPsi
  (plotManhattan), 32
plotExpression (plotManhattan), 32
plotFilterExpression (plotManhattan), 32
plotFilterVariability (plotManhattan), 32
plotFunctions (plotManhattan), 32
plotManhattan, FraserDataSet-method (plotManhattan), 32
plotQQ (plotManhattan), 32
plotQQ, FraserDataSet-method (plotManhattan), 32
plotSpliceMetricRank (plotManhattan), 32
plotVolcano (plotManhattan), 32
plotVolcano, FraserDataSet-method (plotManhattan), 32
potentialImpactAnnotations, 42
predictedMeans
(getter_setter_functions), 21
pseudocount (getter_setter_functions), 21
psiTypes, 44
pVals (getter_setter_functions), 21
pVals, (getter_setter_functions), 21
results, FraserDataSet-method, 45
rho (getter_setter_functions), 21
rho, (getter_setter_functions), 21
samples, 47
samples, FraserDataSet-method (samples), 47
samples<-(samples), 47
samples<-, FraserDataSet-method (samples), 47
saveFraserDataSet (loadFraserDataSet), 27
scanBamParam (samples), 47
scanBamParam, FraserDataSet-method (samples), 47
scanBamParam<-(samples), 47
scanBamParam<-, FraserDataSet-method (samples), 47
strandSpecific (samples), 47
strandSpecific, FraserDataSet-method (samples), 47
strandSpecific<-(samples), 47
strandSpecific<-, FraserDataSet-method (samples), 47
subset.FRASER, 51
summarizeJunctions, 9
useEnsembl, 4
verbose (getter_setter_functions), 21
verbose<-(getter_setter_functions), 21
workingDir (samples), 47
workingDir, FraserDataSet-method (samples), 47
workingDir<-(samples), 47
workingDir<-, FraserDataSet-method (samples), 47
zScore, (getter_setter_functions), 21
zScores (getter_setter_functions), 21