Package ‘FRASER’

October 17, 2020

Type Package

Title Find RAre Splicing Events in RNA-Seq Data

Version 1.0.2

Date 2020-08-27

Description Detection of rare aberrant splicing events in transcriptome profiles. The workflow aims to assist the 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

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

RoxygenNote 7.1.1

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, GenomeInfoDb, GenomicAlignments, GenomicFeatures, GenomicRanges, IRanges, grDevices, ggplot2, ggrepel, HDF5Array, matrixStats, methods, pcaMethods, pheatmap, plotly, PRROC, RColorBrewer, rhdf5, Rsubread, R.utils, S4Vectors, stats, tibble, tools, utils, VGAM

Suggests BiocStyle, knitr, rmarkdown, testthat, covr, TxDb.Hsapiens.UCSC.hg19.knownGene, org.Hs.eg.db,

LinkingTo Rcpp, RcppArmadillo

Collate variables.R getNSetterFuns.R FRASER-package.R FraserDataSet-class.R AllGenerics.R Fraser-pipeline.R accessor-methods.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 saveHDF5Objects.R RcppExports.R autoencoder.R updateD.R updateE.R updateRho.R pvalsNzscore.R makeSimulatedDataset.R fitCorrectionMethods.R plotMethods.R zzz.R

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

git_branch RELEASE_3_11

git_last_commit 0b8a9a0

git_last_commit_date 2020-09-04

Date/Publication 2020-10-16

Author Christian Mertes [aut, cre],
Ines Scheller [aut],
Vicente Yepez [ctb],
Julien Gagneur [aut]

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

R topics documented:

annotateRanges 2
assayNames,FraserDataSet-method 4
assays,FraserDataSet-method 4
calculatePSIValues 5
countRNA 6
createTestFraserSettings 10
filtering 10
FRASER 12
FraserDataSet 15
FraserDataSet-class 16
getter_setter_functions 17
injectOutliers 19
K 20
length,FraserDataSet-method 21
loadFraserDataSet 22
makeSimulatedFraserDataSet 23
optimHyperParams 24
plotFunctions 25
psiTypes 30
results 31
samples 33
subset.FRASER 36

Index **37**

`annotateRanges` *Annotates the given FraserDataSet with the HGNC symbol with biomaRt*

Description

Annotates the given FraserDataSet with the HGNC symbol with biomaRt

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
)
```

Arguments

`fds`	FraserDataSet
`feature`	Defines which feature (default is HGNC symbol) should be annotated.
`featureName`	Name of the feature in the FraserDataSet mcols.
`biotype`	The biotype.
`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 type of gene IDs in the `TxDb` object (see AnnotationDbi::keytypes(orgDb) 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 If this is NULL, then the default one is used, currently this is `org.Hs.eg.db`.

Value

FraserDataSet

Examples

```
fds <- createTestFraserDataSet()

## Not run:
### Two ways to annotage ranges with gene names:
# either using biomart:
fds <- annotateRanges(fds, GRCh=NULL)
rowRanges(fds, type="psi5")[,"hgnc_symbol"]
```

```
# or with a 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="psi5")[,"hgnc_symbol"]

## End(Not run)
```

assayNames,FraserDataSet-method

Returns the assayNames of FRASER

Description

Returns the assayNames of FRASER

Usage

```
## 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

```
## 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
```

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.

`calculatePSIValues` *PSI value calculation*

Description

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

Usage

```
calculatePSIValues(
  fds,
  types = psiTypes,
  overwriteCts = FALSE,
  BPPARAM = bpparam()
)
```

Arguments

`fds`	A FraserDataSet object
`types`	A vector with the psi types which should be calculated. Default is all of psi5, psi3 and psiSite.
`overwriteCts`	FALSE or TRUE (the default) the total counts (aka N) will be recalculated based on the existing junction counts (aka K)
`BPPARAM`	the BiocParallel parameters for the parallelization

Value

FraserDataSet

Examples

```
fds <- createTestFraserDataSet()
fds <- calculatePSIValues(fds, types="psi5")

### 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

```
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))),
  ...
)

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")
)

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
)

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
)
```

Arguments

`fds`	A FraserDataSet object
`NcpuPerSample`	A BiocParallel param object or a positive integer to configure the parallel back-end of the internal loop per sample
`minAnchor`	Minimum overlap around the Donor/Acceptor for non spliced reads. Default to 5
`recount`	if TRUE the cache is ignored and the bam file is recounted.
`BPPARAM`	the BiocParallel parameters for the parallelization
`genome`	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.

`junctionMap`	A object or file containing a map of all junctions of interest across all samples
`filter`	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.
`minExpressionInOneSample`	The minimal split read count in at least one sample that is required for an intron to pass the filter.
`keepNonStandardChromosomes`	Logical value indicating if non standard chromosomes should also be counted. Defaults to `TRUE`.
`countDir`	The directory in which the tsv containing the position and counts of the junctions should be placed.
`...`	Further parameters passed on to Rsubread::featureCounts.
`countFiles`	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.
`outDir`	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.
`splitCountRanges`	The merged GRanges object containing the positions of all the introns in the dataset over all samples.
`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.
`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.

getSplitReadCountsForAllSamples counts split reads for all samples and getNonSplitReadCountsForAllSamples counts non split reads overlapping splice sites for all samples. addCountsToFraserDataSet 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.

getSplitReadCountsForAllSamples returns a GRanges object.

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

```
# 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 functionallity of the FRASER package. Dependent on the request only the sample annotation or a full fitted model is returned.

Usage

```
createTestFraserSettings(workingDir = tempdir())

createTestFraserDataSet(workingDir = tempdir(), rerun = FALSE)
```

Arguments

`workingDir`	directory where to store HDF5 and RDS files. Defaults to the current tempory R session folder.
`rerun`	Defaults to `FALSE`. If set to `TRUE` it reruns the full fit of the model.

Value

a FraserDataSet object which contains a test case

Examples

```
fds <- createTestFraserSettings()
fds

fds <- createTestFraserDataSet()
fds
```

`filtering` *Filtering FraserDataSets*

Description

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

Usage

```
filterExpressionAndVariability(
  fds,
  minExpressionInOneSample = 20,
  quantile = 0.05,
  quantileMinExpression = 1,
  minDeltaPsi = 0,
  filter = TRUE,
  delayed = ifelse(ncol(fds) <= 300, FALSE, TRUE),
  BPPARAM = bpparam()
)

filterExpression(
  fds,
  minExpressionInOneSample = 20,
  quantile = 0.05,
  quantileMinExpression = 1,
  filter = TRUE,
  delayed = ifelse(ncol(fds) <= 300, FALSE, TRUE),
  BPPARAM = bpparam()
)

filterVariability(
  fds,
  minDeltaPsi = 0,
  filter = TRUE,
  delayed = ifelse(ncol(fds) <= 300, FALSE, TRUE),
  BPPARAM = bpparam()
)
```

Arguments

`fds`	A FraserDataSet object
`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.
`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 functions filters out both introns with low read support and introns that are not variable across samples.
- `filterExpression`: This function filters out introns and corresponding splice sites that have low read support in all samples.
- `filterVariability`: This function filters out introns and corresponding splice sites which do not show variablity across samples.

Examples

```
fds <- createTestFraserDataSet()
fds <- filterExpressionAndVariability(fds)
```

`FRASER` *FRASER: Find RAre Splicing Events in RNA-seq data*

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

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

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")

fit(
  fds,
  implementation = c("PCA", "PCA-BB-Decoder", "AE", "AE-weighted", "PCA-BB-full",
   "fullAE", "PCA-regression", "PCA-reg-full", "PCA-BB-Decoder-no-weights", "BB"),
  q,
  type = "psi3",
  rhoRange = c(1e-08, 1 - 1e-08),
  weighted = FALSE,
  noiseAlpha = 1,
  convergence = 1e-05,
  iterations = 15,
  initialize = TRUE,
  control = list(),
  BPPARAM = bpparam(),
  nSubset = 15000,
  verbose = FALSE,
  minDeltaPsi = 0.1
)
```

Arguments

`fds`	A FraserDataSet object
`q`	The encoding dimensions to be used during the fitting proceadure. Should be fitted using `optimHyperParams` if unknown. If a named vector is provided it is used for the different splicing types.
`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.
`...`	Additional parameters passed on to the internal fit function
`type`	The type of PSI (psi5, psi3 or psiSite for theta/splicing efficiency)
`logit`	Indicates if z scores are computed on the logit scale (default) or in the natural (psi) 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.
`rhoRange`	Defines the range of values that rho parameter from the beta-binomial distribution 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.

`initialize`	If FALSE and a fit has been previoulsy 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().
`nSubset`	The size of the subset to be used in fitting if subsetting is used.
`verbose`	Controls the level of information printed during the fit.
`minDeltaPsi`	Minimal delta psi of an intron to be be considered a variable intron.

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 distrbution 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 already 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.
- `fit`: This method correct for confounders in the data and fits a beta-binomial distribution to the introns.

Author(s)

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

Examples

```
# 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())
}

# preprocessing
fds <- createTestFraserDataSet()

### when running FRASER on a real dataset, one should run the following
### two commands first (not run here to make the example run faster):
# 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="psi5")

### To directly run the nomial and adjusted p value and z score
### calculation, the following functions can be used:
# fds <- calculatePvalues(fds, type="psi5")
# head(pVals(fds, type="psi5"))
# fds <- calculatePadjValues(fds, type="psi5", method="BY")
# head(padjVals(fds, type="psi5"))
# fds <- calculateZscore(fds, type="psi5")
# head(zScores(fds, type="psi5"))
```

`FraserDataSet` *The FRASER dataset object*

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 GRange 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` *FraserDataSet*

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 *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", dist = "BetaBinomial", ...)

padjVals(fds, type = currentType(fds), dist = c("BetaBinomial"), ...)

predictedMeans(fds, type = currentType(fds))

deltaPsiValue(fds, type = currentType(fds))

currentType(fds)

currentType(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 psiSite)
`value`	The new value to be assigned.

`byGroup`	If TRUE, aggregation by donor/acceptor site will be done.
`...`	Internally used parameteres.
`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.
`dist`	Distribution for which the p-values should be extracted.
`all`	Logical value indicating whether `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<-`: To remove certain junctions from being used in the train step of the encoding dimension we can set the `featureExclusion` vector to `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.
- `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.
- `currentType<-`: Sets the psi type that is to be 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 opimization 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 opimization was not run yet.
- `dontWriteHDF5`: Gets the current value of whether the assays should be stored as hdf5 files.
- `dontWriteHDF5<-`: Sets whether the assays should be stored as hdf5 files.
- `verbose`: Dependend on the level of verbosity the algorithm reports more or less to the user. 0 means being quiet and 10 means everything.
- `verbose<-`: 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) <- "psi5"
currentType(fds)

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

# get statistics
pVals(fds)
padjVals(fds)
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="psiSite") <- sample(
        c(FALSE, TRUE), nrow(mcols(fds, type="psiSite")), replace=TRUE)
featureExclusionMask(fds, type="psiSite")

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

`injectOutliers` *Inject artificial outliers in an existing fds*

Description

Inject artificial outliers in an existing fds

Usage

```
injectOutliers(
  fds,
  type = c("psi5", "psi3", "psiSite"),
  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 <- 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 = NULL, side = c("ofInterest", "otherSide"))

## S4 replacement method for signature 'FraserDataSet'
counts(object, type = NULL, 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`	An integer matrix containing the counts.

Value

FraserDataSet

Examples

```
fds <- createTestFraserDataSet()

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

`length,FraserDataSet-method`

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

Description

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

Usage

```
## 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

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

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

Value

FraserDataSet

Examples

```
fds <- createTestFraserSettings()
name(fds) <- "saveing_test"

# make sure the object is saved to disc
dontWriteHDF5(fds) <- FALSE
fdsSaved <- saveFraserDataSet(fds)
fdsSaved

# load object from disc
fdsLoaded <- loadFraserDataSet(dir=workingDir(fds), name=name(fds))
fdsLoaded

all.equal(fdsSaved, fdsLoaded)
```

`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 specificed sample size and injection method
fds2 <- makeSimulatedFraserDataSet(m=10, j=100, q=3)
fds2
```

`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,
  implementation = "PCA",
  q_param = seq(2, min(40, ncol(fds)), by = 3),
  noise_param = 0,
  minDeltaPsi = 0.1,
  iterations = 5,
  setSubset = 15000,
  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 (psi5, psi3 or psiSite 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
`noise_param`	Vector specifying which noise levels should be tested.
`minDeltaPsi`	Minimal delta psi of an intron to be 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`	A BiocParallel object to run the computation in parallel
`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

Examples

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

# run hyperparameter optimization
fds <- optimHyperParams(fds, type="psi5", q_param=c(2, 5))

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

`plotFunctions` *Visualization functions for FRASER*

Description

The FRASER package provides mutliple 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 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

```
plotVolcano(
  fds,
  sampleID,
  type = c("psi3", "psi5", "psiSite"),
  basePlot = TRUE,
  aggregate = FALSE,
  main = NULL,
  label = NULL,
  deltaPsiCutoff = 0.3,
  padjCutoff = 0.1,
  ...
)

plotAberrantPerSample(
  fds,
  main,
  type = c("psi3", "psi5", "psiSite"),
  padjCutoff = 0.1,
  zScoreCutoff = NA,
  deltaPsiCutoff = 0.3,
  aggregated = TRUE,
  BPPARAM = bpparam(),
  ...
)

plotExpression(
  fds,
  type = c("psi5", "psi3", "psiSite"),
  site = NULL,
  result = NULL,
  colGroup = NULL,
  basePlot = TRUE,
  main = NULL,
  label = "aberrant",
  ...
)

plotExpectedVsObservedPsi(
  fds,
  type = c("psi5", "psi3", "psiSite"),
  idx = NULL,
  result = NULL,
  colGroup = NULL,
  main = NULL,
  basePlot = TRUE,
  label = "aberrant",
  ...
)

plotQQ(
  fds,
  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()),
  ...
)

plotEncDimSearch(
  fds,
  type = c("psi3", "psi5", "psiSite"),
  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
)

plotCountCorHeatmap(
  fds,
  type = c("psi5", "psi3", "psiSite"),
  logit = FALSE,
  topN = 50000,
  topJ = 5000,
  minMedian = 1,
  main = NULL,
  normalized = FALSE,
  show_rownames = FALSE,
  show_colnames = FALSE,
  minDeltaPsi = 0.1,
  annotation_col = NA,
  annotation_row = NA,
  border_color = NA,
  nClust = 5,
  plotType = c("sampleCorrelation", "junctionSample"),
  sampleClustering = NULL,
  plotMeanPsi = TRUE,
  plotCov = TRUE,
  ...
)
```

Arguments

`fds`	An FraserDataSet object.

`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 psiSite (for SE).
`basePlot`	if `TRUE` (default), use the R base plot version, else use the plotly framework.
`aggregate, aggregated`	If TRUE, the pvalues are aggregated by gene, otherwise junction level pvalues are used (default).
`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 sampleIDs.
`padjCutoff, zScoreCutoff, deltaPsiCutoff`	Significance, Z-score or delta psi cutoff to mark outliers
`...`	Additional parameters passed to plot() or plot_ly() if not stated otherwise in the details for each plot function
`BPPARAM`	BiocParallel parameter to use.
`result`	The result table to be used by the method.
`colGroup`	Group of samples that should be colored.
`idx, site`	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.
`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`	Character string indicating the type of correlation heatmap that should be plotted. 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 variability filter. Defaults to FALSE.
`onlyExpressedIntrons`	Logical value indicating whether to show only introns that also pass the expression filter. Defaults to FALSE.
`logit`	If TRUE, the default, psi values are plotted in logit space.
`topN, topJ`	Top x most variable junctions that should be used in the heatmap. TopN is used for sample-sample correlation heatmaps and topJ for junction-sample correlation heatmaps.
`minMedian, minDeltaPsi`	Minimal median value or minimal delta psi 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 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.

Details

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

- plotAberrantPerSample()
- plotVolcano()
- plotExpression()
- plotQQ()
- plotExpectedVsObservedPsi()
- plotCountCorHeatmap()
- plotFilterExpression()
- plotFilterVariability()
- plotEncDimSearch()

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 (psi5, psi3 or SE (psiSite)) 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.

`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.

Value

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

Examples

```
fds <- createTestFraserDataSet()

plotEncDimSearch(fds, type="psi5")
plotAberrantPerSample(fds, padjCutoff=NA, zScoreCutoff=0.5)
plotVolcano(fds, "sample1", "psi5")

# for this example a padj cutoff of 1 is used to get results for the small
# example dataset and be able to show the usage of the plot functions
res <- results(fds, padjCutoff=1, zScoreCutoff=NA, deltaPsiCutoff=NA)
res
plotExpression(fds, result=res[2], type="psi5")
plotQQ(fds, result=res[1])
plotExpectedVsObservedPsi(fds, type="psi5", idx=5)

plotCountCorHeatmap(fds, "psiSite")
```

`psiTypes` *Available psi types*

Description

Available psi types

Usage

```
psiTypes
```

Format

An object of class `character` of length 3.

Examples

```
  # to show available psi types:
  psiTypes
```

results *Extracting results and aberrant splicing events*

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

```
results(x, ...)

## S4 method for signature 'FraserDataSet'
results(
  x,
  sampleIDs = samples(x),
  padjCutoff = 0.05,
  zScoreCutoff = NA,
  deltaPsiCutoff = 0.3,
  minCount = 5,
  psiType = c("psi3", "psi5", "psiSite"),
  additionalColumns = NULL,
  BPPARAM = bpparam()
)

resultsByGenes(res, geneColumn = "hgncSymbol", method = "BY")

aberrant(
  fds,
  type = currentType(fds),
  padjCutoff = 0.05,
  deltaPsiCutoff = 0.3,
  zScoreCutoff = NA,
  minCount = 5,
  by = c("none", "sample", "feature"),
  aggregate = FALSE,
  ...
)
```

Arguments

`x, fds`	FraserDataSet
`...`	Further arguments can be passed to the method. If "zscores", "padjVals" or "dPsi" is given, the values of those arguments are used to define the aberrant events.
`sampleIDs`	A vector of sample IDs for which results should be retrieved
`padjCutoff`	The FDR cutoff to be applied or NA if not requested.
`zScoreCutoff`	The z-score cutoff to be applied or NA if not requested.
`deltaPsiCutoff`	The cutoff on delta psi or NA if not requested.

`minCount`	The minimum count value of the total coverage of an intron to be considered as significant. result
`psiType`	The psi types for which the results 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.
`res`	Result as created with `results()`
`geneColumn`	The name of the column in `mcols(res)` that contains the gene symbols.
`method`	The p.adjust method that is being used to adjust p values per sample.
`type`	Splicing type (psi5, psi3 or psiSite)
`by`	By default none which means no grouping. But if `sample` or `feature` is specified the sum by sample or feature is returned
`aggregate`	If TRUE the returned object is based on the grouped features

Value

GRanges object

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 vaule of "by"

Examples

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

# extract results: for this example dataset, z score cutoff of 2 is used to
# get at least one result and show the output
res <- results(fds, padjCutoff=NA, zScoreCutoff=3, deltaPsiCutoff=0.05)
res

# aggregate the results by genes (gene symbols need to be annotated first
# using annotateRanges() function)
resultsByGenes(res)

# get aberrant events per sample: on the example data, nothing is aberrant
# based on the adjusted p-value
aberrant(fds, type="psi5", by="sample")

# get aberrant events per gene (first annotate gene symbols)
fds <- annotateRangesWithTxDb(fds)
aberrant(fds, type="psi5", by="feature", zScoreCutoff=2, padjCutoff=NA,
        aggregate=TRUE)

# find aberrant junctions/splice sites
aberrant(fds, type="psi5")
```

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.

Usage

```
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, ...)

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 psiSite)
`...`	Further parameters. For mapSeqLevels: further parameters passed to GenomeInfoDb::mapSeqlevels().
`fds`	FraserDataSet
`style`	The style of the chromosome names.

Details

`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.

Value

Getter method return the respective current value.

Author(s)

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

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

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

Examples

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

---

subset.FRASER — *Subsetting by indices for junctions*

Description

Providing subsetting by indices through the single-bracket operator

Usage

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

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

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) definig if we subset by junctions or splice sites

Value

A subsetted `FraserDataSet` object

Examples

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

Index

∗ **datasets**
psiTypes, 30
[,FraserDataSet,ANY,ANY,ANY-method (subset.FRASER), 36
‘featureExclusionMask<-‘ (getter_setter_functions), 17

aberrant, *29*
aberrant (results), 31
addCountsToFraserDataSet, *9*
addCountsToFraserDataSet (countRNA), 6
annotateRanges, 2
annotateRangesWithTxDb (annotateRanges), 2
assayNames,FraserDataSet-method, 4
assays,FraserDataSet-method, 4
assays<-,FraserDataSet,DelayedMatrix-method (assays,FraserDataSet-method), 4
assays<-,FraserDataSet,list-method (assays,FraserDataSet-method), 4
assays<-,FraserDataSet,SimpleList-method (assays,FraserDataSet-method), 4
available.genomes, *7*

bamFile (samples), 33
bamFile,FraserDataSet-method (samples), 33
bamFile<- (samples), 33
bamFile<-,FraserDataSet-method (samples), 33
bestQ (getter_setter_functions), 17

calculatePadjValues (FRASER), 12
calculatePSIValues, 5
calculatePvalues (FRASER), 12
calculateZscore (FRASER), 12
condition (samples), 33
condition,FraserDataSet-method (samples), 33
condition<- (samples), 33
condition<-,FraserDataSet-method (samples), 33
countNonSplicedReads, *9*
countNonSplicedReads (countRNA), 6
countRNA, 6
countRNAData, *9*
countRNAData (countRNA), 6
counts,FraserDataSet-method (K), 20
counts<-,FraserDataSet-method (K), 20
countSplitReads, *9*
countSplitReads (countRNA), 6
createTestFraserDataSet (createTestFraserSettings), 10
createTestFraserSettings, 10
currentType (getter_setter_functions), 17
currentType<- (getter_setter_functions), 17

deltaPsiValue (getter_setter_functions), 17
dontWriteHDF5 (getter_setter_functions), 17
dontWriteHDF5<- (getter_setter_functions), 17

featureExclusionMask (getter_setter_functions), 17
featureExclusionMask, (getter_setter_functions), 17
featureExclusionMask<- (getter_setter_functions), 17
filterExpression (filtering), 10
filterExpressionAndVariability (filtering), 10
filtering, 10
filterVariability (filtering), 10
fit (FRASER), 12
FRASER, 12
FRASER.mcols.get (samples), 33
FRASER.rowRanges.get (samples), 33
FraserDataSet, 15, *16*
FraserDataSet-class, 16

getBSgenome, *7*
getNonSplitReadCountsForAllSamples, *9*
getNonSplitReadCountsForAllSamples (countRNA), 6
getSplitReadCountsForAllSamples, *9*
getSplitReadCountsForAllSamples (countRNA), 6
getter_setter_functions, 17

hyperParams (getter_setter_functions), 17

injectOutliers, 19

K, 20

length,FraserDataSet-method, 21
loadFraserDataSet, 22

makeSimulatedFraserDataSet, 23
mapSeqlevels (samples), 33
mergeCounts, *9*
mergeCounts (countRNA), 6

N (K), 20
name (samples), 33
name,FraserDataSet-method (samples), 33
name<- (samples), 33
name<-,FraserDataSet-method (samples), 33
nonSplicedReads (samples), 33
nonSplicedReads,FraserDataSet-method (samples), 33
nonSplicedReads<- (samples), 33
nonSplicedReads<-,FraserDataSet-method (samples), 33

optimHyperParams, *13*, 24

padjVals (getter_setter_functions), 17
padjVals, (getter_setter_functions), 17
pairedEnd (samples), 33
pairedEnd,FraserDataSet-method (samples), 33
pairedEnd<- (samples), 33
pairedEnd<-,FraserDataSet-method (samples), 33
pheatmap, *29*
plotAberrantPerSample (plotFunctions), 25
plotCountCorHeatmap (plotFunctions), 25
plotEncDimSearch (plotFunctions), 25
plotExpectedVsObservedPsi (plotFunctions), 25
plotExpression (plotFunctions), 25
plotFilterExpression (plotFunctions), 25
plotFilterVariability (plotFunctions), 25
plotFunctions, 25
plotQQ (plotFunctions), 25
plotVolcano (plotFunctions), 25
predictedMeans (getter_setter_functions), 17
pseudocount (getter_setter_functions), 17
psiTypes, 30
pVals (getter_setter_functions), 17
pVals, (getter_setter_functions), 17

results, 31
results,FraserDataSet-method (results), 31
resultsByGenes (results), 31
rho (getter_setter_functions), 17
rho, (getter_setter_functions), 17

samples, 33
samples,FraserDataSet-method (samples), 33
samples<- (samples), 33
samples<-,FraserDataSet-method (samples), 33
saveFraserDataSet (loadFraserDataSet), 22
scanBamParam (samples), 33
scanBamParam,FraserDataSet-method (samples), 33
scanBamParam<- (samples), 33
scanBamParam<-,FraserDataSet-method (samples), 33
strandSpecific (samples), 33
strandSpecific,FraserDataSet-method (samples), 33
strandSpecific<- (samples), 33
strandSpecific<-,FraserDataSet-method (samples), 33
subset.FRASER, 36
summarizeJunctions, *8*

verbose (getter_setter_functions), 17
verbose<- (getter_setter_functions), 17

workingDir (samples), 33
workingDir,FraserDataSet-method (samples), 33
workingDir<- (samples), 33
workingDir<-,FraserDataSet-method (samples), 33

zScore, *(getter_setter_functions)*, 17
zScores *(getter_setter_functions)*, 17