# Package ‘regsplice’

January 24, 2026

**Version** 1.37.0

**Title** L1-regularization based methods for detection of differential splicing

**Description** Statistical methods for detection of differential splicing (differential exon usage) in RNA-seq and exon microarray data, using L1-regularization (lasso) to improve power.

**Author** Lukas M. Weber [aut, cre]

**Maintainer** Lukas M. Weber `<lmweb012@gmail.com>`

**URL** `https://github.com/lmweber/regsplice`

**BugReports** `https://github.com/lmweber/regsplice/issues`

**License** MIT + file LICENSE

**LazyData** true

**Imports** glmnet, SummarizedExperiment, S4Vectors, limma, edgeR, stats, pbapply, utils, methods

**Suggests** testthat, BiocStyle, knitr, rmarkdown

**VignetteBuilder** knitr

**RoxygenNote** 7.1.0

**biocViews** ImmunoOncology, AlternativeSplicing, DifferentialExpression, DifferentialSplicing, Sequencing, RNASeq, Microarray, ExonArray, ExperimentalDesign, Software

**Collate** 'class_RegspliceResults.R' 'class_RegspliceData.R' 'LRTests.R' 'createDesignMatrix.R' 'filterLowCounts.R' 'filterZeros.R' 'fitting_functions_multiple_genes.R' 'fitting_functions_single_gene.R' 'helper_functions.R' 'initializeResults.R' 'regsplice_wrapper.R' 'runNormalization.R' 'runVoom.R' 'summaryTable.R'

**git_url** https://git.bioconductor.org/packages/regsplice

**git_branch** devel

**git_last_commit** e903be0

**git_last_commit_date** 2025-10-29

**Repository** Bioconductor 3.23
**Date/Publication** 2026-01-23

# Contents

## `createDesignMatrix` *Create design matrix.*

### Description

Create a model design matrix for a single gene.

### Usage

```
createDesignMatrix(condition, n_exons)
```

### Arguments

| | |
|---|---|
| `condition` | Experimental conditions for each sample (character or numeric vector, or factor). |
| `n_exons` | Number of exons in the gene (integer). |

### Details

Creates a model design matrix for a single gene in the format required by the `regsplice` model fitting functions. Required inputs are the experimental conditions (groups) for each sample, and the number of exons in the gene.

The design matrix includes main effect terms for each exon and each sample, and interaction terms between the exons and conditions.

Note that the design matrix does not include main effect terms for the conditions, since these are absorbed into the main effect terms for the samples. In addition, the design matrix does not include an intercept column, since it is simpler to let the model fitting functions add an intercept term later.

The model fitting functions in subsequent steps call this function once for each gene.

### Value

Returns a model design matrix for the gene, in the format required by the `regsplice` model fitting functions.

### See Also

`fitRegMultiple` `fitNullMultiple` `fitFullMultiple` `LRTests`

### Examples

```
condition <- rep(c(0, 1), each = 3)
n_exons <- 10
X <- createDesignMatrix(condition, n_exons)
```

---

## `filterLowCounts` *Filter low-count exons.*

---

### Description

Filter low-count exons from RNA-seq read count data.

### Usage

```
filterLowCounts(rs_data, filter_min_per_exon = 6, filter_min_per_sample = 3)
```

### Arguments

| | |
|---|---|
| `rs_data` | `RegspliceData` object. |
| `filter_min_per_exon` | Filtering parameter: minimum number of reads per exon bin, summed across all biological samples. Default is 6. |
| `filter_min_per_sample` | Filtering parameter: minimum number of reads per biological sample; i.e. for each exon bin, at least one sample must have this number of reads. Default is 3. |

### Details

Filters low-count exon bins from RNA-seq read count data. Any remaining single-exon genes (after filtering) are also removed (since differential splicing requires multiple exon bins).

Input data is assumed to be in the form of a `RegspliceData` object. See `RegspliceData` for details.

The arguments `filter_min_per_exon` and `filter_min_per_sample` control the amount of filtering. Exon bins that meet the filtering conditions are kept. Default values for the arguments are provided; however, these should be adjusted depending on the total number of samples and the number of samples per condition.

After filtering low-count exon bins, any remaining genes containing only a single exon bin are also removed (since differential splicing requires multiple exon bins).

Filtering should be skipped when using exon microarray data. (When using the `regsplice` wrapper function, filtering can be disabled with the argument `filter = FALSE`).

Previous step: Filter zero-count exon bins with `filterZeros`. Next step: Calculate normalization factors with `runNormalization`.

### Value

Returns a `RegspliceData` object.

### See Also

`filterZeros` `runNormalization`

### Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data <- filterZeros(rs_data)
rs_data <- filterLowCounts(rs_data)
```

## `filterZeros` *Filter zero-count exons.*

### Description

Filter exons with zero RNA-seq read counts in all biological samples.

### Usage

```
filterZeros(rs_data)
```

### Arguments

| | |
|---|---|
| `rs_data` | `RegspliceData` object. |

## Details

Removes exon bins with zero RNA-seq read counts in all biological samples. Any remaining single-exon genes (after filtering) are also removed (since differential splicing requires multiple exon bins).

Input data is assumed to be in the form of a `RegspliceData` object. See `RegspliceData` for details.

After filtering zero-count exon bins, any remaining genes containing only a single exon bin are also removed (since differential splicing requires multiple exon bins).

Filtering should be skipped when using exon microarray data. (When using the `regsplice` wrapper function, filtering can be disabled with the argument `filter = FALSE`).

Previous step: Create `RegspliceData` object with `RegspliceData` constructor function. Next step: Filter low-count exon bins with `filterLowCounts`.

## Value

Returns a `RegspliceData` object.

## See Also

`RegspliceData` `filterLowCounts`

## Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data <- filterZeros(rs_data)
```

---

`fitRegMultiple` *Fit models.*

---

## Description

Model fitting functions for `regsplice` package.

## Usage

```
fitRegMultiple(
  rs_results,
  rs_data,
  alpha = 1,
  lambda_choice = c("lambda.min", "lambda.1se"),
  seed = NULL,
  ...
)

fitNullMultiple(rs_results, rs_data, seed = NULL, ...)

fitFullMultiple(rs_results, rs_data, seed = NULL, ...)
```

## Arguments

| | |
|---|---|
| `rs_results` | `RegspliceResults` object, which will be used to store results. Initialized using the constructor function `RegspliceResults()`. See `RegspliceResults` for details. |
| `rs_data` | `RegspliceData` object. In the case of RNA-seq read count data, this has been pre-transformed with `runVoom`. Contains `counts` and `weights` data matrices, and vector of experimental conditions for each biological sample stored in `colData`. See `RegspliceData` for details. |
| `alpha` | Elastic net parameter `alpha` for `glmnet` model fitting functions. Must be between 0 (ridge regression) and 1 (lasso). Default is 1 (lasso). See `glmnet` documentation for more details. |
| `lambda_choice` | Parameter to select which optimal `lambda` value to choose from the `cv.glmnet` cross validation fit. Choices are "lambda.min" (model with minimum cross-validated error) and "lambda.1se" (most regularized model with cross-validated error within one standard error of minimum). Default is "lambda.min". See `glmnet` documentation for more details. |
| `seed` | Random seed (integer). Default is NULL. Provide an integer value to set the random seed for reproducible results. |
| `...` | Other arguments to pass to `cv.glmnet`, `glmnet`, or `glm`. |

## Details

There are three model fitting functions:

`fitRegMultiple` fits regularized (lasso) models containing an optimal subset of exon:condition interaction terms for each gene. The model fitting procedure penalizes the interaction terms only, so that the main effect terms for exons and samples are always included. This ensures that the null model is nested, allowing likelihood ratio tests to be calculated.

`fitNullMultiple` fits the null models, which do not contain any interaction terms.

`fitFullMultiple` fits full models, which contain all exon:condition interaction terms for each gene.

See `createDesignMatrix` for more details about the terms in each model.

The fitting functions fit models for all genes in the data set.

A random seed can be provided with the seed argument, to generate reproducible results.

If the rs_data object does not contain a weights matrix, all exon bins are weighted equally.

Previous step: Initialize RegspliceResults object with initializeResults. Next step: Calculate likelihood ratio tests with LRTests.

## Value

Returns a RegspliceResults object containing deviance and degrees of freedom of the fitted models. See RegspliceResults for details.

## See Also

createDesignMatrix RegspliceResults initializeResults LRTests

glmnet cv.glmnet glm

## Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data <- filterZeros(rs_data)
rs_data <- filterLowCounts(rs_data)
rs_data <- runNormalization(rs_data)
rs_data <- runVoom(rs_data)

rs_results <- initializeResults(rs_data)

rs_results <- fitRegMultiple(rs_results, rs_data)
rs_results <- fitNullMultiple(rs_results, rs_data)
rs_results <- fitFullMultiple(rs_results, rs_data)
```

---

## initializeResults *Initialize RegspliceResults object.*

---

## Description

Initialize a RegspliceResults object, which will contain the results of the regsplice analysis.

## Usage

```
initializeResults(rs_data)
```

## Arguments

| | |
|---|---|
| `rs_data` | `RegspliceData` object. This should contain gene IDs in a column named `gene_IDs` in the row meta-data, which can be accessed with the accessor function `rowData`. |

## Details

Creates a `RegspliceResults` object containing gene names only. This object will subsequently be populated using the functions `fitRegMultiple`, `fitNullMultiple`, `fitFullMultiple`, and `LRTests`.

Previous step: Calculate `limma-voom` transformation and weights with `runVoom`. Next step: Fit models with `fitRegMultiple`, `fitNullMultiple`, and `fitFullMultiple`.

## Value

Returns a `RegspliceResults` object containing gene IDs only.

## See Also

`RegspliceData` `RegspliceResults` `fitRegMultiple` `fitNullMultiple` `fitFullMultiple` `LRTests` `summaryTable`

## Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data <- filterZeros(rs_data)
rs_data <- filterLowCounts(rs_data)
rs_data <- runNormalization(rs_data)
rs_data <- runVoom(rs_data)

rs_results <- initializeResults(rs_data)
```

## LRTests *Calculate likelihood ratio tests.*

### Description

Calculate likelihood ratio tests between fitted models and null models.

### Usage

```
LRTests(rs_results, when_null_selected = c("ones", "full", "NA"))
```

### Arguments

| | |
|---|---|
| `rs_results` | `RegspliceResults` object containing results generated by `fitRegMultiple`, `fitNullMultiple`, and `fitFullMultiple`. If `when_null_selected = "ones"` or `"NA"`, the "full" models are not required. See `RegspliceResults` for details. |
| `when_null_selected` | Which option to use for genes where the lasso model selects zero interaction terms, i.e. identical to the null model. Options are `"ones"`, `"full"`, and `"NA"`. Default is `"ones"`. See below for details. |

### Details

The regularized (lasso) fitted models contain an optimal subset of exon:condition interaction terms for each gene, and the "full" fitted models contain all exon:condition interaction terms. The null models contain zero interaction terms, so they are nested within the fitted models.

The likelihood ratio (LR) tests compare the fitted models against the nested null models.

If the regularized (lasso) model contains at least one exon:condition interaction term, the LR test compares the lasso model against the null model. However, if the lasso model contains zero interaction terms, then the lasso and null models are identical, so the LR test cannot be calculated. The `when_null_selected` argument lets the user choose what to do in these cases: either set p-values equal to 1 (`when_null_selected = "ones"`); or calculate a LR test using the "full" model containing all exon:condition interaction terms (`when_null_selected = "full"`), which reduces power due to the larger number of terms, but allows the evidence for differential exon usage among these genes to be distinguished. You can also return `NA`s for these genes (`when_null_selected = "NA"`).

The default option is `when_null_selected = "ones"`. This simply calls all these genes non-significant, which in most cases is sufficient since we are more interested in genes with strong evidence for differential exon usage. However, if it is important to rank the low-evidence genes in your data set, use the `when_null_selected = "full"` option.

If `when_null_selected = "ones"` or `when_null_selected = "NA"`, the "full" fitted models are not required.

Previous step: Fit models with `fitRegMultiple`, `fitNullMultiple`, and `fitFullMultiple`. Next step: Generate summary table of results with `summaryTable`.

## Value

Returns a `RegspliceResults` object containing results of the LR tests. The results consist of the following entries for each gene:

- p_vals: raw p-values
- p_adj: multiple testing adjusted p-values (Benjamini-Hochberg false discovery rates, FDR)
- LR_stats: likelihood ratio test statistics
- df_tests: degrees of freedom of likelihood ratio tests

## See Also

`RegspliceResults` `initializeResults` `fitRegMultiple` `fitNullMultiple` `fitFullMultiple` `summaryTable`

## Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data <- filterZeros(rs_data)
rs_data <- filterLowCounts(rs_data)
rs_data <- runNormalization(rs_data)
rs_data <- runVoom(rs_data)

rs_results <- initializeResults(rs_data)

rs_results <- fitRegMultiple(rs_results, rs_data)
rs_results <- fitNullMultiple(rs_results, rs_data)
rs_results <- fitFullMultiple(rs_results, rs_data)

rs_results <- LRTests(rs_results)
```

---

`regsplice` *Wrapper function to run regsplice.*

---

## Description

Wrapper function to run a `regsplice` analysis with a single command.

## Usage

```
regsplice(
  rs_data,
  filter_zeros = TRUE,
  filter_low_counts = TRUE,
  filter_min_per_exon = 6,
  filter_min_per_sample = 3,
  normalize = TRUE,
  norm_method = "TMM",
  voom = TRUE,
  alpha = 1,
  lambda_choice = c("lambda.min", "lambda.1se"),
  when_null_selected = c("ones", "full", "NA"),
  seed = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `rs_data` | `RegspliceData` object containing input data. See `RegspliceData` for details. |
| `filter_zeros` | Whether to filter zero-count exon bins, using `filterZeros`. Default is TRUE. Set to FALSE for exon microarray data. |
| `filter_low_counts` | Whether to filter low-count exon bins, using `filterLowCounts`. Default is TRUE. Set to FALSE for exon microarray data. |
| `filter_min_per_exon` | Filtering parameter for low-count exon bins: minimum number of reads per exon bin, summed across all biological samples. Default is 6. See `filterLowCounts` for details. |
| `filter_min_per_sample` | Filtering parameter for low-count exon bins: minimum number of reads per biological sample; i.e. for each exon bin, at least one sample must have this number of reads. Default is 3. See `filterLowCounts` for details. |
| `normalize` | Whether to calculate normalization factors, using `runNormalization`. Default is TRUE. If FALSE, non-normalized library sizes will be used. Set to FALSE for exon microarray data. |
| `norm_method` | Normalization method to use. Options are `"TMM"`, `"RLE"`, `"upperquartile"`, and `"none"`. Default is `"TMM"`. See `runNormalization` for details. |
| `voom` | Whether to calculate `limma-voom` transformation and weights, using `runVoom`. Default is TRUE. If FALSE, model fitting functions will use the raw input data (not recommended for count data) with exon bins weighted equally. Set to FALSE for exon microarray data. |
| `alpha` | Elastic net parameter `alpha` for `glmnet` model fitting functions. Must be between 0 (ridge regression) and 1 (lasso). Default is 1 (lasso). See `glmnet` documentation for more details. |

| | |
|---|---|
| `lambda_choice` | Parameter to select which optimal `lambda` value to choose from the `cv.glmnet` cross validation fit. Choices are "lambda.min" (model with minimum cross-validated error) and "lambda.1se" (most regularized model with cross-validated error within one standard error of minimum). Default is "lambda.min". See `glmnet` documentation for more details. |
| `when_null_selected` | Which option to use for genes where the lasso model selects zero interaction terms, i.e. identical to the null model. Options are `"ones"`, `"full"`, and `"NA"`. Default is `"ones"`. See `LRTests` for details. |
| `seed` | Random seed (integer). Default is NULL. Provide an integer value to set the random seed for reproducible results. |
| `...` | Other arguments to pass to `cv.glmnet`, `glmnet`, or `glm`. |

## Details

This wrapper function runs the `regsplice` analysis pipeline with a single command.

The required input format is a `RegspliceData` object, which is created with the `RegspliceData` constructor function.

The wrapper function calls each of the individual functions in the analysis pipeline in sequence. You can also run the individual functions directly, which provides additional flexibility and insight into the statistical methodology. See the vignette for a description of the individual functions and an example workflow.

After running the analysis pipeline, a summary table of the results can be displayed with `summaryTable`.

Note that when using exon microarray data, the filtering, normalization, and `voom` steps should be disabled with the respective arguments.

See `RegspliceData` for details on constructing the input data object; `filterZeros` and `filterLowCounts` for details about filtering; `runNormalization` and `runVoom` for details about calculation of normalization factors and `voom` transformation and weights; `createDesignMatrix` for details about the model design matrices; `fitRegMultiple`, `fitNullMultiple`, or `fitFullMultiple` for details about the model fitting functions; and `LRTests` for details about the likelihood ratio tests.

## Value

Returns a `RegspliceResults` object containing fitted model results and likelihood ratio (LR) test results. The LR test results consist of the following entries for each gene:

- p_vals: raw p-values
- p_adj: multiple testing adjusted p-values (Benjamini-Hochberg false discovery rates, FDR)
- LR_stats: likelihood ratio test statistics
- df_tests: degrees of freedom of likelihood ratio tests

## See Also

`RegspliceData` `RegspliceResults` `initializeResults` `filterZeros` `filterLowCounts` `runNormalization` `runVoom` `createDesignMatrix` `fitRegMultiple` `fitNullMultiple` `fitFullMultiple` `LRTests` `summaryTable`

### Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_results <- regsplice(rs_data)

summaryTable(rs_results)
```

---

## `RegspliceData-class` *RegspliceData objects.*

---

### Description

`RegspliceData` objects contain data in the format required by functions in the `regsplice` analysis pipeline.

### Usage

```
RegspliceData(counts, gene_IDs = NULL, n_exons = NULL, condition = NULL)

## S4 method for signature 'RegspliceData'
assays(x, withDimnames, ..., value)

countsData(x)

## S4 method for signature 'RegspliceData'
countsData(x)

weightsData(x)

## S4 method for signature 'RegspliceData'
weightsData(x)

## S4 method for signature 'RegspliceData'
rowData(x)

## S4 method for signature 'RegspliceData'
colData(x, ..., value)
```

```
## S4 method for signature 'RegspliceData,ANY,ANY,ANY'
x[i, j]
```

## Arguments

| | |
|---|---|
| `counts` | RNA-seq read counts or exon microarray intensities (matrix or data frame). Rows are exons, and columns are biological samples. Alternatively, `counts` also accepts a `SummarizedExperiment` input object containing all required input data, which may be useful when running `regsplice` as part of a pipeline with other packages. |
| `gene_IDs` | Vector of gene IDs (character vector). Length is equal to the number of genes. |
| `n_exons` | Vector of exon lengths (numeric vector of integers), i.e. the number of exon bins per gene. Length is equal to the number of genes. |
| `condition` | Experimental condition for each biological sample (character or numeric vector, or factor). |
| `x` | `RegspliceData` object (for accessor or subsetting functions). |
| `withDimnames` | See `SummarizedExperiment::assays()`. |
| `...` | Additional arguments for replacement with `` `[<-` ``. |
| `value` | Value for replacement with `` `[<-` ``. |
| `i` | Gene names (character vector) or row numbers (numeric vector) for subsetting genes or exons. Note that when subsetting whole genes, gene names (character vector) should be provided instead of row numbers, to avoid possible errors due to selecting incorrect row numbers. Row numbers may be provided to subset individual exons. |
| `j` | Column numbers (numeric vector) for subsetting biological samples. |

## Details

The `RegspliceData` format is based on the `SummarizedExperiment` container. Initially, objects contain raw data along with meta-data for rows (genes and exons) and columns (biological samples). During subsequent steps in the `regsplice` analysis pipeline, the data values are modified, and additional data and meta-data are added to the object. Final results are stored in a `RegspliceResults` object.

`RegspliceData` objects are created with the constructor function `RegspliceData()`.

Required inputs for the constructor function are `counts` (matrix or data frame of RNA-seq read counts or exon microarray intensities), `gene_IDs` (vector of gene IDs), `n_exons` (vector of exon lengths, i.e. number of exon bins per gene), and `condition` (vector of experimental conditions for each biological sample).

Alternatively, the inputs can be provided as a `SummarizedExperiment` object, which will be parsed to extract each of these components. This may be useful when running `regsplice` as part of a pipeline together with other packages.

See the vignette for an example showing how to construct `gene_IDs` and `n_exons` from a column of gene:exon IDs.

Exon microarray intensities should be log2-transformed, which is usually done during pre-processing of microarray data. (RNA-seq counts will be transformed automatically during the `regsplice` analysis pipeline; see `runVoom`.)

After creating a `RegspliceData` object, the wrapper function `regsplice` can be used to run the analysis pipeline with a single command. Alternatively, you can run the individual functions for each step in the pipeline, beginning with `filterZeros` (see vignette for details).

## Value

Returns a `RegspliceData` object.

## Fields

`counts` Matrix of RNA-seq read counts or exon microarray intensities. Rows are exons, and columns are biological samples.

`weights` (Optional) Matrix of observation-level weights. Rows are exons, and columns are biological samples. Created by the `runVoom` function.

`rowData` `DataFrame` of row meta-data. This should contain two columns: `gene_IDs` and `exon_IDs`, which are created by the `RegspliceData` constructor function.

`colData` `DataFrame` of column meta-data. This contains the experimental condition and (optionally) normalization factors for each biological sample. Normalization factors are created by the `runVoom` function.

## Accessor functions

- `countsData()`: Accesses the `counts` data matrix.
- `weightsData()`: Accesses the (optional) `weights` data matrix.
- `rowData()`: Accesses the `DataFrame` of row meta-data. This should contain two columns: `gene_IDs` and `exon_IDs`.
- `colData()`: Accesses the `DataFrame` of column meta-data. This contains the experimental condition and (optionally) normalization factors for each biological sample.

## Subsetting

Subsetting of `RegspliceData` objects is performed with square brackets, `x[i, j]`, where `x` is the name of the object. The subsetting operations are designed to keep data and meta-data in sync.

For subsetting by rows, there are two possibilities:

- Subsetting genes: To subset whole genes, provide a character vector of gene names to the argument `i`. The returned object will contain all rows corresponding to these genes. Row numbers should not be used when subsetting whole genes, since this risks potential errors due to selecting incorrect rows.
- Subsetting exons: To subset individual exons, provide the corresponding row numbers to the argument `i`.

For subsetting by columns (biological samples), provide the corresponding column numbers to the argument `j`.

## See Also

regsplice filterZeros

## Examples

```
# ---------
# Example 1
# ---------

counts <- matrix(sample(100:200, 14 * 6, replace = TRUE), nrow = 14, ncol = 6)
gene_IDs <- paste0("gene", 1:5)
n_exons <- c(3, 2, 3, 1, 5)
condition <- rep(c(0, 1), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data
countsData(rs_data)
rowData(rs_data)
colData(rs_data)

rs_data[1, ]
rs_data[1, 1:3]

rs_data["gene1", ]
rs_data["gene1", 1:3]


# --------------------
# Example 2 (Vignette)
# --------------------

file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data
head(countsData(rs_data))
rowData(rs_data)
colData(rs_data)

rs_data[1, ]
rs_data[1, 1:3]
```

```
rs_data["ENSG00000000003", ]
rs_data["ENSG00000000003", 1:3]
```

---

`RegspliceResults-class` *RegspliceResults objects.*

---

## Description

`RegspliceResults` objects contain the results of a `regsplice` analysis.

## Usage

```
RegspliceResults(gene_IDs)

gene_IDs(x)

## S4 method for signature 'RegspliceResults'
gene_IDs(x)

p_vals(x)

## S4 method for signature 'RegspliceResults'
p_vals(x)

p_adj(x)

## S4 method for signature 'RegspliceResults'
p_adj(x)

LR_stats(x)

## S4 method for signature 'RegspliceResults'
LR_stats(x)

df_tests(x)

## S4 method for signature 'RegspliceResults'
df_tests(x)
```

## Arguments

| | |
|---|---|
| `gene_IDs` | Gene identifiers or names (character vector). |
| `x` | `RegspliceResults` object (for accessor functions). |

## Details

RegspliceResults objects are created with the constructor function RegspliceResults(), which requires the gene IDs as an argument.

Once created, RegspliceResults objects are then populated using the functions fitRegMultiple, fitNullMultiple, fitFullMultiple, and LRTests.

The function summaryTable can be used to display a summary table of the results.

## Value

Returns an empty RegspliceResults object.

## Fields

gene_IDs Gene identifiers or names (character vector).

fit_reg_dev Deviance of fitted regularized (lasso) models from fitRegMultiple.

fit_reg_df Degrees of freedom of fitted regularized (lasso) models from fitRegMultiple.

fit_null_dev Deviance of fitted null models from fitNullMultiple.

fit_null_df Degrees of freedom of fitted null models from fitNullMultiple.

fit_full_dev Deviance of fitted full models from fitFullMultiple.

fit_full_df Degrees of freedom of fitted full models from fitFullMultiple.

p_vals Raw p-values (numeric vector).

p_adj Multiple testing adjusted p-values (Benjamini-Hochberg false discovery rates, FDR).

LR_stats Likelihood ratio test statistics.

df_tests Degrees of freedom of likelihood ratio tests.

## Accessor functions

- gene_IDs(): Accesses gene identifiers or names.
- p_vals(): Accesses raw p-values.
- p_adj(): Accesses multiple testing adjusted p-values (Benjamini-Hochberg false discovery rates, FDR).
- LR_stats(): Accesses likelihood ratio test statistics.
- df_tests(): Accesses degrees of freedom of likelihood ratio tests.

## See Also

fitRegMultiple fitNullMultiple fitFullMultiple LRTests summaryTable

## Examples

```
# initialize RegspliceResults object
gene_IDs <- paste0("gene", 1:5)
RegspliceResults(gene_IDs)
```

## runNormalization *Calculate normalization factors.*

### Description

Calculate normalization factors to scale library sizes, using the TMM (trimmed mean of M-values) method implemented in `edgeR`.

### Usage

```
runNormalization(rs_data, norm_method = "TMM")
```

### Arguments

| | |
|---|---|
| `rs_data` | `RegspliceData` object, which has already been filtered with `filterZeros` and `filterLowCounts`. |
| `norm_method` | Normalization method to use. Options are `"TMM"`, `"RLE"`, `"upperquartile"`, and `"none"`. See documentation for `calcNormFactors` in `edgeR` package for details. Default is `"TMM"`. |

### Details

Normalization factors are used to scale the raw library sizes (total read counts per sample). We use the TMM (trimmed mean of M-values) normalization method (Robinson and Oshlack, 2010), as implemented in the `edgeR` package.

For more details, see the documentation for `calcNormFactors` in the `edgeR` package.

This step should be performed after filtering with `filterZeros` and `filterLowCounts`. The normalization factors are then used by `limma-voom` in the next step (`runVoom`).

The normalization factors are stored in a new column named `norm_factors` in the column meta-data (`colData` slot) of the `RegspliceData` object. The `colData` can be accessed with the accessor function `colData()`.

Normalization should be skipped when using exon microarray data. (When using the `regsplice` wrapper function, normalization can be disabled with the argument `normalize = FALSE`).

Previous step: Filter low-count exon bins with `filterLowCounts`. Next step: Calculate `limma-voom` transformation and weights with `runVoom`.

### Value

Returns a `RegspliceData` object. Normalization factors are stored in the column `norm_factors` in the column meta-data (`colData` slot), which can be accessed with the `colData()` accessor function.

### See Also

`filterLowCounts` `runVoom`

## Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data <- filterZeros(rs_data)
rs_data <- filterLowCounts(rs_data)
rs_data <- runNormalization(rs_data)
```

---

`runVoom` *Calculate 'voom' transformation and weights.*

---

## Description

Use `limma-voom` to transform counts and calculate exon-level weights.

## Usage

```
runVoom(rs_data)
```

## Arguments

| | |
|---|---|
| `rs_data` | `RegspliceData` object, which has been filtered with `filterZeros` and `filterLowCounts`, and (optionally) normalization factors added with `runNormalization`. |

## Details

Raw counts do not fulfill the statistical assumptions required for linear modeling. The `limma-voom` methodology transforms counts to log2-counts per million (logCPM), and calculates exon-level weights based on the observed mean-variance relationship. Linear modeling methods can then be applied.

For more details, see the documentation for `voom` in the `limma` package.

Note that `voom` assumes that exon bins (rows) with zero or low counts have already been removed, so this step should be done after filtering with `filterZeros` and `filterLowCounts`.

Normalization factors can be provided in a column named `norm_factors` in the column meta-data (`colData` slot) of the `RegspliceData` object. These will be used by `voom` to calculate normalized library sizes. If normalization factors are not provided, `voom` will use non-normalized library sizes (columnwise total counts) instead.

The experimental conditions or group labels for each biological sample are assumed to be in a column named condition in the column meta-data (colData slot) of the RegspliceData object. This column is created when the object is initialized with the RegspliceData() constructor function.

The transformed counts are stored in the updated counts matrix, which can be accessed with the countsData accessor function. The weights are stored in a new data matrix labeled weights, which can be accessed with the weightsData accessor function. In addition, the normalized library sizes (if available) are stored in a new column named lib_sizes in the column meta-data (colData slot).

If you are using exon microarray data, this step should be skipped, since exon microarray intensities are already on a continuous scale.

Previous step: Calculate normalization factors with runNormalization. Next step: Initialize RegspliceResults object with the constructor function RegspliceResults().

## Value

Returns a RegspliceData object. Transformed counts are stored in the counts matrix, and weights are stored in a new weights data matrix. The data matrices can be accessed with the accessor functions countsData and weightsData.

## See Also

runNormalization fitRegMultiple fitNullMultiple fitFullMultiple

## Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_data <- filterZeros(rs_data)
rs_data <- filterLowCounts(rs_data)
rs_data <- runNormalization(rs_data)
rs_data <- runVoom(rs_data)
```

---

## summaryTable *Summary table.*

---

## Description

Display summary table of results from a regsplice analysis.

## Usage

```
summaryTable(
  rs_results,
  n = 20,
  threshold = 0.05,
  rank_by = c("FDR", "p-value", "none")
)
```

## Arguments

| | |
|---|---|
| `rs_results` | `RegspliceResults` object containing results of a `regsplice` analysis, generated with wrapper function `regsplice` (or individual functions up to `LRTests`). See `RegspliceResults` for details. |
| `n` | Number of genes to display in summary table. Default is 20. If the total number of significant genes up to the significance threshold is less than `n`, only the significant genes are shown. Set to `Inf` to display all significant genes; or set both `n = Inf` and `threshold = 1` to display all genes in the data set. |
| `threshold` | Significance threshold (for either FDR or raw p-values, depending on choice of argument `rank_by`). Default is 0.05. Set to 1 to display all `n` genes; or set both `n = Inf` and `threshold = 1` to display all genes in the data set. |
| `rank_by` | Whether to rank genes by false discovery rate (FDR), raw p-values, or no ranking. Choices are `"FDR"`, `"p-value"`, and `"none"`. Default is `"FDR"`. |

## Details

The results of a `regsplice` analysis consist of a set of multiple testing adjusted p-values (Benjamini-Hochberg false discovery rates, FDR) quantifying the statistical evidence for differential exon usage (DEU) for each gene. Typically, the adjusted p-values are used to rank the genes in the data set according to their evidence for DEU, and an appropriate significance threshold (e.g. FDR < 0.05) can be used to generate a list of genes with statistically significant evidence for DEU.

The main `regsplice` functions return results in the form of a `RegspliceResults` object, which contains slots for gene names, fitted model results, raw p-values, multiple testing adjusted p-values (Benjamini-Hochberg FDR), likelihood ratio (LR) test statistics, and degrees of freedom of the LR tests. See `RegspliceResults` and the main `regsplice` wrapper function `regsplice` for details.

This function generates a summary table of the results. The results are displayed as a data frame of the top `n` most highly significant genes, ranked according to either FDR or raw p-values, up to a specified significance threshold (e.g. FDR < 0.05).

The argument `rank_by` controls whether to rank by FDR or raw p-values. The default is to rank by FDR.

To display results for all genes up to the significance threshold, set the argument `n = Inf`. To display results for all genes in the data set, set both `n = Inf` and `threshold = 1`.

Previous step: Run `regsplice` pipeline with the `regsplice` wrapper function (or individual functions up to `LRTests`).

## Value

Returns a data frame containing results for the top n most highly significant genes, up to the specified significance threshold for the FDR or raw p-values.

## See Also

RegspliceResults regsplice

## Examples

```
file_counts <- system.file("extdata/vignette_counts.txt", package = "regsplice")
data <- read.table(file_counts, header = TRUE, sep = "\t", stringsAsFactors = FALSE)
head(data)

counts <- data[, 2:7]
tbl_exons <- table(sapply(strsplit(data$exon, ":"), function(s) s[[1]]))
gene_IDs <- names(tbl_exons)
n_exons <- unname(tbl_exons)
condition <- rep(c("untreated", "treated"), each = 3)

rs_data <- RegspliceData(counts, gene_IDs, n_exons, condition)

rs_results <- regsplice(rs_data)

summaryTable(rs_results)
summaryTable(rs_results, n = Inf, threshold = 1)
```

# Index