Package: splikit (via r-universe)

May 13, 2026

Title Analysing RNA Splicing in Single-Cell RNA Sequencing Data

Version 2.3.1

Description Provides analysis of high-dimensional single-cell splicing data. Offers a framework to extract and work with ratio-based data structures derived from single-cell RNA sequencing experiments. Provides both a modern 'R6' object-oriented interface and direct matrix manipulation functions. Core functionalities are implemented in 'C++' via 'Rcpp' to ensure high performance and scalability on large datasets.

URL https://csglab.github.io/splikit/, https://github.com/csglab/splikit

BugReports https://github.com/csglab/splikit/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports Matrix, data.table, methods, stats, Rcpp, R6

LinkingTo Rcpp, RcppArmadillo

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2

VignetteBuilder knitr

Config/testthat/edition 3

Depends R (>= 4.1.0)

NeedsCompilation yes

Author Arsham Mikaeili Namini [aut, cre] (ORCID: <https://orcid.org/0000-0002-9453-6951>)

Maintainer Arsham Mikaeili Namini <arsham.mikaeilinamini@mail.mcgill.ca>

Repository https://cran.r-universe.dev

Date/Publication 2026-05-13 10:48:18 UTC

RemoteUrl https://github.com/cran/splikit

RemoteRef HEAD

RemoteSha 464cc6753bc0787f1b810ffb9a7fdb7a98dcc840

Contents

`find_variable_events` *Calculate the Sum Deviance for Inclusion and Exclusion Matrices*

Description

Calculate the Sum Deviance for Inclusion and Exclusion Matrices

Usage

```
find_variable_events(
  m1_matrix,
  m2_matrix = NULL,
  min_row_sum = 50,
  n_threads = 1,
  verbose = FALSE,
  ...
)
```

Arguments

m1_matrix	A matrix representing the inclusion matrix. Rows are events, columns are barcodes.
m2_matrix	A matrix representing the exclusion matrix. Rows are events, columns are barcodes.
min_row_sum	A numeric value specifying the minimum row sum threshold for filtering events. Defaults to 50.
n_threads	If the module OpenPM is available for your device, the function suggests using multi-thread processing for even faster computation.
verbose	Logical. If TRUE, prints progress and informational messages. Default is FALSE.
...	Additional arguments to be passed.

Value

A data.table containing the events and their corresponding sum deviance values.

Examples

```
# loading the toy dataset
toy_obj <- load_toy_M1_M2_object()

# getting HVE (high variable events)
 HVE <- find_variable_events(toy_obj$m1, toy_obj$m2)

 # printing the results
 print(HVE[order(-sum_deviance)])
```

---

find_variable_genes *Find Variable Genes Using Variance or Deviance-Based Metrics*

Description

Identifies highly variable genes from a sparse gene expression matrix using one of two methods: variance-stabilizing transformation (VST) or deviance-based modeling. The VST method uses a C++-accelerated approach to compute standardized variance, while the deviance-based method models gene variability across libraries using negative binomial deviances.

Usage

```
find_variable_genes(
  gene_expression_matrix,
  method = "vst",
  n_threads = 1,
  verbose = FALSE,
  ...
)
```

Arguments

gene_expression_matrix	A sparse gene expression matrix (of class Matrix) with gene names as row names.
method	Character string, either "vst" or "sum_deviance". The default is "sum_deviance". "vst" uses a variance-stabilizing transformation to identify variable genes. "sum_deviance" computes per-library deviances and combines them with a row variance metric.
n_threads	If OpenMP is available for your device, the function suggests using multi-thread processing for even faster computation (only for sum_deviance method).
verbose	Logical. If TRUE, prints progress and informational messages. Default is FALSE.
...	Additional arguments (currently unused).

Value

A data.table containing gene names (column events) and computed metrics. For the deviance method, this includes sum_deviance and variance columns.

Examples

```
library(Matrix)
# loading the toy dataset
toy_obj <- load_toy_M1_M2_object()

# getting high variable genes
HVG_VST <- find_variable_genes(toy_obj$gene_expression, method = "vst")
# sum_deviance method
HVG_DEV <- find_variable_genes(toy_obj$gene_expression, method = "sum_deviance")

# Using multi-threading for faster computation (sum_deviance method only)
HVG_DEV_MT <- find_variable_genes(toy_obj$gene_expression,
                                  method = "sum_deviance",
                                  n_threads = 4) # 4 threads

# printing the results
print(HVG_VST[order(-standardize_variance)])
print(HVG_DEV[order(-sum_deviance)])
```

---

get_pseudo_correlation

Compute Pseudo-Correlation Using Beta-Binomial Model

Description

This function calculates a pseudo R^2-like correlation metric using a beta-binomial model implemented in C++. It takes in a data matrix ZDB_matrix and two model matrices for inclusion and exclusion, respectively. The function now supports both sparse and dense matrices for m1 and m2, and allows selection between Cox-Snell and Nagelkerke R^2 metrics.

Usage

```
get_pseudo_correlation(
  ZDB_matrix,
  m1_inclusion = NULL,
  m2_exclusion = NULL,
  metric = "CoxSnell",
  suppress_warnings = TRUE,
  verbose = FALSE
)
```

Arguments

ZDB_matrix	A numeric dense matrix of shape (events x samples). Should have rownames representing events.
m1_inclusion	A numeric matrix (dense or sparse) of the same number of rows as ZDB_matrix, representing inclusion features.
m2_exclusion	A numeric matrix (dense or sparse) of the same number of rows as ZDB_matrix, representing exclusion features.
metric	Character string specifying which R^2 metric to compute. Options are "CoxSnell" (default) or "Nagelkerke".
suppress_warnings	Logical. If TRUE (default), suppresses warnings during computation (e.g., due to ill-conditioned inputs).
verbose	Logical. If TRUE, prints progress and informational messages. Default is FALSE.

Value

A data.table with the following columns:

event The event names from ZDB_matrix rownames.

pseudo_correlation The computed pseudo R^2 correlation values using the specified metric.

null_distribution Null correlation values from a permuted version of ZDB_matrix.

Examples

```
set.seed(42)
# get the m1 object
junction_abundance_object <- load_toy_SJ_object()
m1_obj <- make_m1(junction_ab_object = junction_abundance_object)

# obtaining the m1 and eventdata
m1_inclusion <- m1_obj$m1_inclusion_matrix
eventdata <- m1_obj$event_data
m2_exclusion <- make_m2(m1_inclusion, eventdata)

# creating a dummy ZDB
ZDB_matrix <- matrix(rnorm(n = (nrow(m1_inclusion) * ncol(m1_inclusion)), sd = 7),
nrow = nrow(m1_inclusion),
```

```
ncol = ncol(m1_inclusion))
rownames(ZDB_matrix) <- rownames(m1_inclusion)

# m1 and m2 can now be either sparse or dense matrices
# Example with dense matrices (backward compatible)
m1_dense <- as.matrix(m1_inclusion)
m2_dense <- as.matrix(m2_exclusion)
pseudo_r_square_cox <- get_pseudo_correlation(ZDB_matrix, m1_dense, m2_dense)
print(pseudo_r_square_cox)

# Example with sparse matrices (more memory efficient)
pseudo_r_square_sparse <- get_pseudo_correlation(ZDB_matrix, m1_inclusion, m2_exclusion)

# Example using Nagelkerke R-squared instead of Cox-Snell
pseudo_r_square_nagel <- get_pseudo_correlation(ZDB_matrix, m1_inclusion, m2_exclusion,
                                                metric = "Nagelkerke")
print(pseudo_r_square_nagel)
```

`get_rowVar` *Calculate Row-wise Variance for Dense or Sparse Matrices*

Description

Efficiently computes the variance of each row for either a base R dense matrix or a sparse dgCMatrix, via a single Rcpp entry point. Logs progress messages to the R console.

Usage

```
get_rowVar(M, verbose = FALSE)
```

Arguments

`M`	A numeric matrix (base R matrix) or a sparse matrix of class `"dgCMatrix"`.
`verbose`	Logical. If `TRUE`, prints progress and informational messages. Default is `FALSE`.

Details

Dispatches in C++ between dense and sparse implementations to avoid unnecessary overhead or external dependencies. Uses compressed-column traversal for sparse inputs.

Value

A numeric vector of length `nrow(M)` containing the variance of each row.

Note

Only 32-bit integer indices are supported, due to limitations in R's internal matrix representations. This function will not work with matrices that exceed the 32-bit integer indexing range.

Examples

```
library(Matrix)
# Dense example
dm <- matrix(rnorm(1000), nrow = 100)
get_rowVar(dm)
# Sparse example
sm <- rsparsematrix(100, 10, density = 0.1)
get_rowVar(sm)
```

`get_silhouette_mean` *Compute Average Silhouette Width with Logging*

Description

Computes the average silhouette width for a clustering solution using Euclidean distance.

Usage

```
get_silhouette_mean(X, cluster_assignments, n_threads = 1, verbose = FALSE)
```

Arguments

`X`	A numeric matrix where rows are observations and columns are features.
`cluster_assignments`	An integer vector of cluster assignments, which must be the same length as the number of rows in `X`.
`n_threads`	Number of threads to use for parallel processing.
`verbose`	Logical. If `TRUE`, prints progress and informational messages. Default is `FALSE`.

Value

A single numeric value: the average silhouette score.

Note

This process can be very slow for large matrices if single-threaded. Use multiple threads to take advantage of parallel computation for significantly faster results.

Examples

```
# Preparing the inputs
set.seed(42)
pc_matrix <- matrix(data = rnorm(n = 10000 * 15, sd = 2), nrow = 10000, ncol = 15)
cluster_numbers <- as.integer(runif(n = 10000, min = 1, max = 10))

# Getting the mean silhouette score
n_threads <- parallel::detectCores()
score <- get_silhouette_mean(pc_matrix, cluster_numbers, n_threads)
print(score)
```

load_toy_M1_M2_object *Load the toy M1/M2 object*

Description

Loads a toy object of M1 and M2 used for examples or testing.

Usage

```
load_toy_M1_M2_object()
```

Value

An R object from the toy_m1_m2_obj.rds file.

load_toy_SJ_object *Load the toy SJ object*

Description

Loads a toy splice junction object used for examples or testing.

Usage

```
load_toy_SJ_object()
```

Value

An R object from the toy_SJ_object.RDS file.

make_eventdata_plus *make_eventdata_plus*

Description

This function reads in a GTF file, extracts gene annotations, and merges them with event-level genomic intervals provided by the user. The final data table contains the original event intervals and the corresponding gene information (for example, `gene_id` and `gene_name`).

Usage

```
make_eventdata_plus(eventdata, GTF_file_direction)
```

Arguments

`eventdata` A data table of genomic intervals for events. Must contain columns:

- `chr`: Chromosome name (e.g., "chr1").
- `start`: Start coordinate of the event.
- `end`: End coordinate of the event.
- `strand`: Numeric strand indicator (`1` or `2`).

`GTF_file_direction`
A character string specifying the path to a GTF file. The file must contain at least these columns: `seqid`, `start`, `end`, `strand`, `gene_id`, and `gene_name`.

Details

1. Read the GTF: Uses `data.table::fread` to load GTF data and convert it to a data table.
2. Subset for Genes: Keeps only rows where `type == "gene"`, retaining columns for chromosome, start, end, strand, gene_id, and gene_name.
3. Strand Conversion: Merges the GTF data with a small lookup table to replace `+` and `-` with numeric strand indicators `1` and `2` (matching STAR).
4. Overlaps: With both data sets keyed, uses `foverlaps()` from `data.table` to find intervals in `eventdata` that fall fully within gene boundaries.

Value

A data table containing overlapping event intervals with added gene metadata (such as `gene_id` and `gene_name`). The columns returned will include both event-level and gene-level information.

`make_gene_count` *make_gene_count*

Description

Constructs sparse gene expression matrices from one or more directories containing 10X Genomics-style output. The function supports barcode filtering using either an external whitelist or the internally provided filtered barcode file.

Usage

```
make_gene_count(
  expression_dirs,
  sample_ids,
  whitelist_barcodes = NULL,
  use_internal_whitelist = TRUE,
  verbose = FALSE
)
```

Arguments

`expression_dirs`
: A character vector or list of strings. Each element must be a path to a directory containing the gene expression matrix files: `matrix.mtx`, `barcodes.tsv`, and `features.tsv` (or `genes.tsv`).

`sample_ids`
: A character vector or list of unique sample identifiers, one for each element in `expression_dirs`. These are used to name outputs in the returned list when multiple samples are provided.

`whitelist_barcodes`
: A list of character vectors. Each list element corresponds to a sample and contains the barcodes to retain for that sample. If `NULL` (default), the function will attempt to use the internal filtered barcode file (e.g., `barcodes.tsv` or `barcodes_filtered.tsv`) if available.

`use_internal_whitelist`
: Logical (default `TRUE`). If `TRUE` and `whitelist_barcodes` is `NULL`, the function will attempt to use the default filtered barcode list from the input directory. If `FALSE`, no internal filtration will be applied unless a whitelist is explicitly provided.

`verbose`
: Logical. If `TRUE`, prints progress and informational messages. Default is `FALSE`.

Details

The function is designed for bulk or single-cell gene expression processing from 10X-style output folders. Each input directory should contain the standard `matrix.mtx`, `features.tsv`/`genes.tsv`, and `barcodes.tsv` files. Barcodes can be filtered using either a provided whitelist or by relying on the `filtered` barcode files output by tools like CellRanger.

If neither an external whitelist nor an internal filtered barcode file is available, all barcodes from the raw matrix will be retained.

Value

If a single sample is provided, returns a sparse matrix of class `"dgCMatrix"` with genes as rows and barcodes as columns. If multiple samples are provided, returns a named list of sparse matrices, one per sample ID.

Dependencies

Requires the **Matrix** package for sparse matrix handling and potentially **data.table** for efficient I/O.

`make_junction_ab` *make_junction_ab*

Description

This function processes STARsolo splicing junction output to create a modality list for splicing data. It supports both single and multiple sample processing, optional barcode filtration, and internal whitelist usage.

Usage

```
make_junction_ab(
  STARsolo_SJ_dirs,
  white_barcode_lists = NULL,
  sample_ids,
  use_internal_whitelist = TRUE,
  verbose = FALSE,
  keep_multi_mapped_junctions = FALSE,
  ...
)
```

Arguments

STARsolo_SJ_dirs	A character vector or list of strings representing the paths to STARsolo SJ directories. Each directory should contain the raw splicing junction output files.
white_barcode_lists	A list of character vectors, each containing barcode whitelist(s) for the corresponding sample. If NULL (default), the function uses the internal STARsolo whitelist if use_internal_whitelist is TRUE.
sample_ids	A character vector or list of unique sample IDs corresponding to each directory in STARsolo_SJ_dirs.
use_internal_whitelist	A logical flag (default TRUE) indicating whether to use the internal STARsolo whitelist located at ../Gene/filtered/barcodes.tsv for each sample when white_barcode_lists is NULL.
verbose	Logical (default FALSE). If TRUE, prints detailed progress messages during processing.
keep_multi_mapped_junctions	Logical (default FALSE). If TRUE, retains multi-mapped junctions (flag 0). If FALSE, only keeps uniquely mapped junctions (flag 1).
...	Additional parameters for future extensions.

Value

A list containing processed splicing modality data for each sample. If a single sample is provided, the function returns the processed data as a list. For multiple samples, a named list is returned. Each sample's result contains:

eventdata A data.table containing feature metadata.

junction_ab A sparse matrix of junction abundance.

make_m1 *make_m1*

Description

This function processes junction abundance data from multiple samples to create a splicing modality inclusion matrix (M1). It merges event data, handles start and end coordinate groups, ensures matrix compatibility, and includes robust error handling.

Usage

```
make_m1(junction_ab_object, min_counts = 1, verbose = FALSE)
```

Arguments

junction_ab_object	A named list where each element represents a sample's junction abundance data. Each element must contain `eventdata` and a sparse matrix.
min_counts	Numeric (default 1). Minimum count threshold for filtering events. Events with total counts below this threshold will be removed.
verbose	Logical (default `FALSE`). If `TRUE`, prints detailed progress messages during processing.

Details

The function requires the following libraries: `data.table`, and `Matrix`.

Value

A list containing the processed data from all samples:

m1_inclusion_matrix A matrix representing the processed inclusion values for all events across all samples.

event_data A `data.table` containing the merged and grouped metadata for each event.

Examples

```
# Example usage
junction_abundance_object <- load_toy_SJ_object()
result <- make_m1(junction_abundance_object)
m1_matrix <- result$m1_inclusion_matrix
event_metadata <- result$event_data
```

make_m2 *make_m2 (Integrated with Automatic Batching)*

Description

Creates the M2 matrix from a given m1_inclusion_matrix and eventdata with intelligent memory management. Automatically detects when the operation would exceed memory limits and switches to a batched sparse matrix approach.

Usage

```
make_m2(
  m1_inclusion_matrix,
  eventdata,
  batch_size = 5000,
  memory_threshold = 2e+09,
  force_fast = FALSE,
  n_threads = 1,
  use_cpp = TRUE,
  verbose = FALSE
)
```

Arguments

m1_inclusion_matrix	A sparse matrix to be modified and used for creating the M2 matrix.
eventdata	A data.table containing event information with at least group_id and an index column.
batch_size	An integer specifying the number of groups to process per batch (default: 5000). Only used when batched processing is triggered.
memory_threshold	A numeric value representing the maximum number of rows allowed in the summary before switching to batched processing (default: 2e9, which is ~93% of 2^31).
force_fast	A logical flag to force fast processing regardless of size estimates (default: FALSE). WARNING: This may cause memory errors on large datasets.
n_threads	Number of threads for parallel processing (default: 1). Used for C++ implementation and batched R processing. Values > 1 require parallel package for batched mode.
use_cpp	Logical flag to use fast C++ implementation (default: TRUE). Falls back to R implementation if FALSE.
verbose	A logical flag for detailed progress reporting (default: FALSE).

Value

A sparse matrix M2 with the dummy row removed and proper adjustments made.

Examples

```
junction_abundance_object <- load_toy_SJ_object()
m1_obj <- make_m1(junction_ab_object = junction_abundance_object)

# obtaining the m1 and eventdata
m1 <- m1_obj$m1_inclusion_matrix
eventdata <- m1_obj$event_data
m2 <- make_m2(m1_inclusion_matrix = m1, eventdata = eventdata)
```

---

make_velo_count *Process Spliced and Unspliced Counts from Velocyto Outputs*

Description

Parses and processes spliced and unspliced gene expression matrices from one or more Velocyto output directories. The function applies barcode filtering using an external whitelist or filtered barcodes file, and optionally merges the results across samples into unified matrices.

Usage

```
make_velo_count(
  velocyto_dirs,
  sample_ids,
  whitelist_barcodes = NULL,
  use_internal_whitelist = TRUE,
  merge_counts = FALSE,
  verbose = FALSE
)
```

Arguments

velocyto_dirs	A character vector or list of strings. Each element should be a path to a Velocyto output directory. Each directory must contain subdirectories (typically filtered or raw) with the required matrix files: spliced.mtx, unspliced.mtx, barcodes.tsv, and genes.tsv or features.tsv.
sample_ids	A character vector or list of unique sample identifiers corresponding to each entry in velocyto_dirs.
whitelist_barcodes	A list of character vectors. Each element should provide a whitelist of barcodes to retain for the corresponding sample. If NULL (default), the function will attempt to use the internally provided filtered barcodes when use_internal_whitelist = TRUE.
use_internal_whitelist	Logical (default TRUE). If TRUE, and whitelist_barcodes is NULL, the function uses the filtered barcode file (if present in the directory). If FALSE, all barcodes from the raw matrix will be used unless a whitelist is explicitly provided.

`merge_counts` Logical (default `FALSE`). If `TRUE`, spliced and unspliced matrices across all samples are merged into two combined matrices (one for spliced, one for unspliced). If `FALSE`, the results are returned per sample.

`verbose` Logical. If `TRUE`, prints progress and informational messages. Default is `FALSE`.

Details

The function assumes that each Velocyto directory follows the 10X-like structure typically produced by tools like Loom or Velocyto CLI. Barcode filtering ensures that only high-quality or selected barcodes are retained for downstream RNA velocity analysis.

When merging matrices, barcodes are prefixed with their corresponding sample ID to avoid collisions and preserve traceability.

Value

A list containing processed gene expression matrices:

- If `merge_counts = FALSE`, returns a named list of sample-specific matrices. Each entry contains:

  `spliced` Sparse matrix of spliced transcript counts.

  `unspliced` Sparse matrix of unspliced transcript counts.

- If `merge_counts = TRUE`, returns a list with two elements:

  `spliced` Merged sparse matrix of spliced counts across all samples.

  `unspliced` Merged sparse matrix of unspliced counts across all samples.

Dependencies

Requires the **Matrix** package for sparse matrix operations and **data.table** for efficient file parsing.

`plot_exclusive_junctions` *Plot transcript-exclusive splice junctions*

Description

Draws a gene-model view of a gene's transcripts (one row per transcript) with splice junctions rendered as arcs above each row. Junctions used by a single transcript of the gene ("transcript-exclusive") are drawn as solid black arcs; shared junctions as thin grey arcs. Works with both Ensembl-style and RefSeq-style GTFs: if a row has no `gene_name` attribute, `gene_id` is used; if no `transcript_name`, `transcript_id` is used.

Usage

```
plot_exclusive_junctions(
  gtf,
  target_gene,
  show_exclusive = TRUE,
  transcript = NULL,
  curvature = -0.2,
  out_file = NULL
)
```

Arguments

gtf	Either a path to an Ensembl- or RefSeq-style GTF, or a pre-loaded data.table of GTF rows.
target_gene	Gene symbol (matches gene_name, or gene_id if no gene_name is present).
show_exclusive	Logical (default TRUE). Restrict to transcripts that own >= 1 exclusive junction.
transcript	Optional character vector of transcript names to pin the plot to (overrides show_exclusive).
curvature	Numeric, arc-height knob for ggplot2::geom_curve().
out_file	Optional character. If given, the plot is also written to this file with ggplot2::ggsave().

Value

An S3 object of class "splikit_junction_plot" (a list) with components:

plot A ggplot object.

info A data.table with one row per drawn (transcript, junction) pair and columns gene_name, gene_id, transcript_name, transcript_id, chr, strand, j_start, j_end, j_width, exclusive, n_tx_with_junction, observed_in_eventdata, row_names_mtx, is_annot. For GTF-only calls the last three are NA.

exons, junctions, tx_order The underlying tables used to build the plot, retained for advanced users.

Printing the object renders the plot and then prints info.

Required packages

Requires the **ggplot2** package (declared in Suggests).

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  # Build a tiny synthetic GTF as a data.table (no external file needed).
  # tx1 has 3 exons (2 junctions); tx2 has 2 exons (1 junction).
  gtf <- data.table::data.table(
    seqname = "chr1",
    source  = "toy",
    type    = rep("exon", 5),
    start   = c(100, 300, 500, 100, 500),
```

```
    end     = c(200, 400, 600, 200, 600),
    score   = ".",
    strand  = "+",
    frame   = ".",
    attr    = c(
      'gene_name "FOO"; transcript_id "tx1"; exon_number "1";',
      'gene_name "FOO"; transcript_id "tx1"; exon_number "2";',
      'gene_name "FOO"; transcript_id "tx1"; exon_number "3";',
      'gene_name "FOO"; transcript_id "tx2"; exon_number "1";',
      'gene_name "FOO"; transcript_id "tx2"; exon_number "2";'
    )
  )
  res <- plot_exclusive_junctions(gtf, "FOO")
  res$info
}
```

`plot_exclusive_junctions_event` *Plot transcript-exclusive splice junctions observed in eventdata*

Description

Sibling of `plot_exclusive_junctions()` that restricts drawn arcs to junctions present in a `splikit` eventdata table. Exon structure and exclusivity are still derived from the GTF, so a black arc means the junction is both transcript-exclusive in the annotation and observed in the data.

Usage

```
plot_exclusive_junctions_event(
  target_gene,
  GTF,
  eventdata,
  show_exclusive = TRUE,
  transcript = NULL,
  curvature = -0.2
)
```

Arguments

`target_gene`	Gene symbol to plot.
`GTF`	Either a GTF path or a pre-loaded `data.table`.
`eventdata`	A `splikit` eventdata `data.table`.
`show_exclusive, transcript, curvature`	See `plot_exclusive_junctions()`.

Details

Works with both Ensembl-style and RefSeq-style GTFs. If `eventdata` lacks a `gene_name` column the function runs `make_eventdata_plus()` internally (requires `GTF` to be a path in that case). Observed junctions not matching any annotated intron are dropped with a message and their count is reported as `novel_junction_count`.

Value

An S3 object of class `"splikit_junction_plot"` with the same structure as in `plot_exclusive_junctions()`, plus `novel_junction_count` for the unannotated observed junctions. The `info$observed_in_eventdata` column is `TRUE` for all rows, and `row_names_mtx` / `is_annot` are populated from `eventdata` when available.

Required packages

Requires the **ggplot2** package (declared in `Suggests`).

---

plot_exclusive_junctions_pdf

Multi-page PDF of transcript-exclusive junctions

Description

Writes a multi-page PDF: page 1 = full gene view (all transcripts, exclusive arcs in black); subsequent pages = one per exclusive-owning transcript, with its exclusive junction in black.

Usage

```
plot_exclusive_junctions_pdf(
  gtf,
  target_gene,
  out_pdf,
  curvature = -0.2,
  width = 10,
  height = NULL
)
```

Arguments

`gtf`, `target_gene`, `curvature`	See `plot_exclusive_junctions()`.
`out_pdf`	Character path for the PDF.
`width`, `height`	PDF page dimensions. `height` = `NULL` (default) sizes to the full-gene page.

Value

Invisibly, a list with `exclusive_transcripts` and `n_pages`.

Required packages

Requires the **ggplot2** package (declared in `Suggests`).

`print.splikit_junction_plot` *Print method for splikit junction-plot results*

Description

Renders the stored plot and then prints the `info` data.table. Called automatically at the REPL when a function like `plot_exclusive_junctions()` is invoked without assignment.

Usage

```
## S3 method for class 'splikit_junction_plot'
print(x, n = NULL, ...)
```

Arguments

`x`	A `splikit_junction_plot` object.
`n`	Rows of `info` to preview. `NULL` (default) shows all.
`...`	Unused.

`splikit` *Create a SplikitObject*

Description

Convenience function to create a SplikitObject.

Usage

```
splikit(...)
```

Arguments

`...`	Arguments passed to `SplikitObject$new()`.

Value

A new SplikitObject instance.

Examples

```
# From existing matrices using the toy dataset
toy <- load_toy_M1_M2_object()
obj <- splikit(m1 = toy$m1, m2 = toy$m2, eventData = toy$eventdata)
```

`SplikitObject` *SplikitObject*

Description

R6 class for splicing analysis in single-cell RNA-seq data. Provides a modern object-oriented interface to splikit functionality while maintaining backward compatibility with existing functions.

Details

The SplikitObject encapsulates the core data structures for splicing analysis:

- `m1`: Inclusion matrix (sparse dgCMatrix)
- `m2`: Exclusion matrix (sparse dgCMatrix)
- `eventData`: Event metadata (data.table)
- `geneExpression`: Optional gene expression matrix

Public fields

`m1` Inclusion matrix (dgCMatrix). Rows are events, columns are cells.

`m2` Exclusion matrix (dgCMatrix). Same dimensions as m1.

`eventData` Event metadata (data.table). One row per event.

`geneExpression` Optional gene expression matrix (dgCMatrix).

`metadata` List containing summary statistics and analysis results.

Methods

Public methods:

- `SplikitObject$new()`
- `SplikitObject$makeM2()`
- `SplikitObject$findVariableEvents()`
- `SplikitObject$findVariableGenes()`
- `SplikitObject$getPseudoCorrelation()`
- `SplikitObject$subset()`
- `SplikitObject$setGeneExpression()`
- `SplikitObject$annotateEvents()`
- `SplikitObject$summary()`

- SplikitObject$print()
- SplikitObject$deepCopy()
- SplikitObject$clone()

Method `new()`**:** Create a new SplikitObject.

Usage:

```
SplikitObject$new(
  junction_ab = NULL,
  m1 = NULL,
  m2 = NULL,
  eventData = NULL,
  min_counts = 1,
  verbose = FALSE
)
```

Arguments:

`junction_ab` A junction abundance object from `make_junction_ab()`. If provided, m1 and eventData are computed automatically.

`m1` An existing inclusion matrix (dgCMatrix).

`m2` An existing exclusion matrix (dgCMatrix).

`eventData` A data.table with event metadata.

`min_counts` Minimum count threshold for filtering events (default: 1).

`verbose` Print progress messages (default: FALSE).

Returns: A new SplikitObject instance.

Method `makeM2()`**:** Compute the M2 exclusion matrix from M1 and eventData.

Usage:

```
SplikitObject$makeM2(
  batch_size = 5000,
  memory_threshold = 2e+09,
  force_fast = FALSE,
  n_threads = 1,
  use_cpp = TRUE,
  verbose = FALSE
)
```

Arguments:

`batch_size` Number of groups per batch for memory management (default: 5000).

`memory_threshold` Maximum rows before switching to batched processing.

`force_fast` Force fast processing regardless of size (default: FALSE).

`n_threads` Number of threads for parallel processing (default: 1).

`use_cpp` Use fast C++ implementation (default: TRUE).

`verbose` Print progress messages (default: FALSE).

Returns: Self (invisibly), for method chaining.

Method `findVariableEvents()`**:** Find highly variable splicing events.

Usage:

```
SplikitObject$findVariableEvents(
  min_row_sum = 50,
  n_threads = 1,
  verbose = FALSE
)
```

Arguments:

`min_row_sum` Minimum row sum threshold for filtering (default: 50).

`n_threads` Number of threads for parallel computation (default: 1).

`verbose` Print progress messages (default: FALSE).

Returns: A data.table with event names and sum_deviance scores.

Method `findVariableGenes()`**:** Find highly variable genes from gene expression data.

Usage:

```
SplikitObject$findVariableGenes(method = "vst", n_threads = 1, verbose = FALSE)
```

Arguments:

`method` Method for variable gene selection: "vst" or "sum_deviance" (default: "vst").

`n_threads` Number of threads for parallel computation (default: 1).

`verbose` Print progress messages (default: FALSE).

Returns: A data.table with gene names and variability scores.

Method `getPseudoCorrelation()`**:** Compute pseudo-correlation between splicing and external data.

Usage:

```
SplikitObject$getPseudoCorrelation(
  ZDB_matrix,
  metric = "CoxSnell",
  suppress_warnings = TRUE
)
```

Arguments:

`ZDB_matrix` Dense matrix of external data (e.g., gene expression PCs). Must have same dimensions as m1.

`metric` R-squared metric: "CoxSnell" or "Nagelkerke" (default: "CoxSnell").

`suppress_warnings` Suppress computation warnings (default: TRUE).

Returns: A data.table with event names, pseudo_correlation, and null_distribution.

Method `subset()`**:** Subset the object by events and/or cells.

Usage:

```
SplikitObject$subset(events = NULL, cells = NULL)
```

Arguments:

`events` Event indices or names to keep.

`cells` Cell indices or names to keep.

Returns: Self (invisibly), for method chaining.

Method `setGeneExpression()`**:** Set the gene expression matrix.

Usage:

```
SplikitObject$setGeneExpression(gene_matrix)
```

Arguments:

`gene_matrix` A gene expression matrix (will be converted to dgCMatrix).

Returns: Self (invisibly), for method chaining.

Method `annotateEvents()`**:** Annotate events with gene information from a GTF file.

Usage:

```
SplikitObject$annotateEvents(GTF_file)
```

Arguments:

`GTF_file` Path to a GTF annotation file.

Returns: Self (invisibly), for method chaining.

Method `summary()`**:** Get a summary of the object.

Usage:

```
SplikitObject$summary()
```

Returns: A list with object statistics.

Method `print()`**:** Print a human-readable summary of the object.

Usage:

```
SplikitObject$print()
```

Method `deepCopy()`**:** Create a deep copy of the object.

Usage:

```
SplikitObject$deepCopy()
```

Arguments:

`deep` If TRUE, creates a deep copy of all data.

Returns: A new SplikitObject with copied data. Validate input matrices and eventData Ensure M2 is computed Ensure matrix is sparse dgCMatrix Standardized error reporting

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SplikitObject$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create from existing matrices using the toy dataset
toy <- load_toy_M1_M2_object()
obj <- SplikitObject$new(m1 = toy$m1, m2 = toy$m2,
                         eventData = toy$eventdata)

# Find variable events
hve <- obj$findVariableEvents(min_row_sum = 50)

# Or build M2 from M1 + eventData and chain into feature selection
obj2 <- SplikitObject$new(m1 = toy$m1, eventData = toy$eventdata)
results <- obj2$makeM2()$findVariableEvents()
```

Index