

Package ‘SpliceWiz’

April 29, 2024

Title interactive analysis and visualization of alternative splicing
in R

Version 1.5.3

Date 2024-03-12

Description The analysis and visualization of alternative splicing (AS) events from RNA sequencing data remains challenging. SpliceWiz is a user-friendly and performance-optimized R package for AS analysis, by processing alignment BAM files to quantify read counts across splice junctions, IRFinder-based intron retention quantitation, and supports novel splicing event identification. We introduce a novel visualization for AS using normalized coverage, thereby allowing visualization of differential AS across conditions. SpliceWiz features a shiny-based GUI facilitating interactive data exploration of results including gene ontology enrichment. It is performance optimized with multi-threaded processing of BAM files and a new COV file format for fast recall of sequencing coverage. Overall, SpliceWiz streamlines AS analysis, enabling reliable identification of functionally relevant AS events for further characterization.

License MIT + file LICENSE

Depends NxtIRFdata

Imports ompBAM, methods, stats, utils, tools, parallel, scales, magrittr, Rcpp (>= 1.0.5), data.table, fst, ggplot2, AnnotationHub, BiocFileCache, BiocGenerics, BiocParallel, Biostrings, BSgenome, DelayedArray, DelayedMatrixStats, genefilter, GenomeInfoDb, GenomicRanges, HDF5Array, htmltools, IRanges, patchwork, pheatmap, progress, plotly, R.utils, rhdf5, rtracklayer, SummarizedExperiment, S4Vectors, shiny, shinyFiles, shinyWidgets, shinydashboard, stringi, rhandsonable, DT, grDevices, heatmaply, matrixStats, RColorBrewer, rvest

Suggests knitr, rmarkdown, crayon, splines, testthat (>= 3.0.0), DESeq2, limma, DoubleExpSeq, edgeR, DBI, GO.db, AnnotationDbi, fgsea, Rsubread

LinkingTo ompBAM, Rcpp, zlibbioc, RcppProgress

SystemRequirements C++11, GNU make

Collate AllImports.R RcppExports.R zzz.R AllClasses.R AllGenerics.R
 ASEFilter-methods.R NxtSE-methods.R globals.R ggplot_themes.R
 example_data.R wrappers.R make_plot_data.R Coverage.R
 covPlotly-methods.R covDataObject-methods.R
 covPlotObject-methods.R plotCoverage.R utils.R File_finders.R
 BuildRef_GO.R BuildRef.R ViewRef.R STAR_utils.R Mappability.R
 ProcessBAM_docs.R ProcessBAM.R CollateData.R MakeSE.R Filters.R
 ASE-methods.R ASE-GLM-edgeR.R dash_filterModules.R
 dash_globals.R dash_settings.R dash_ref_new_ui.R
 dash_ref_new_server.R dash_expr_ui.R dash_expr_server.R
 dash_QC.R dash_filters.R dash_DE_ui.R dash_DE_server.R
 dash_vis_ui.R dash_vis_server.R dash_cov_ui.R dash_cov_server.R
 dash_GO_ui.R dash_GO_server.R dash_ui.R dash_server.R dash.R
 SpliceWiz-package.R

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

VignetteBuilder knitr

biocViews Software, Transcriptomics, RNASeq, AlternativeSplicing,
 Coverage, DifferentialSplicing, DifferentialExpression, GUI,
 Sequencing

URL <https://github.com/alexchwong/SpliceWiz>

BugReports <https://support.bioconductor.org/>

Config/testthat/edition 3

git_url <https://git.bioconductor.org/packages/SpliceWiz>

git_branch devel

git_last_commit 83ccd5e

git_last_commit_date 2024-03-12

Repository Bioconductor 3.19

Date/Publication 2024-04-28

Author Alex Chit Hei Wong [aut, cre, cph],
 Ulf Schmitz [ctb],
 William Ritchie [cph]

Maintainer Alex Chit Hei Wong <a.wong@centenary.org.au>

Contents

SpliceWiz-package	3
ASE-GLM-edgeR	5
ASE-methods	10
ASEFilter-class	17

Build-Reference-methods	20
collateData	28
coord2GR	31
covDataObject-class	32
Coverage	35
covPlotly-class	38
covPlotObject-class	40
example-SpliceWiz-data	46
findSamples	48
Gene-ontology-methods	50
Graphics-User-Interface	54
isCOV	55
makeSE	56
make_plot_data	58
Mappability-methods	60
NxtSE-class	63
plotCoverage	68
processBAM	73
Run_SpliceWiz_Filters	76
setSWthreads	79
STAR-methods	79
theme_white	86
View-Reference-methods	87

Index	90
--------------	-----------

SpliceWiz-package	<i>SpliceWiz: efficient and precise alternative splicing analysis in R</i>
-------------------	--

Description

SpliceWiz is a computationally efficient and user friendly workflow that analyses aligned short-read RNA sequencing for differential intron retention and alternative splicing.

Details

SpliceWiz uses isoform-specific alignments to quantify percent-spliced-in ratios (i.e. ratio of the "included" isoform, as a proportion of "included" and "excluded" isoforms). For intron retention (IR), the abundance of the intron-retaining transcript (included isoform) is quantified using the trimmed-mean depth of intron coverage with reads, whereas the spliced transcript (excluded isoform) is measured as the splicing of the intron as well as that of overlapping introns (since splicing of any overlapping intron implies the intron of interest is not retained). For other forms of alternative splicing, junction reads (reads aligned across splice junctions) are used to quantify included and excluded isoforms.

SpliceWiz processes BAM files (aligned RNA sequencing) using [ompBAM::ompBAM-package](#). ompBAM is a C++ library that allows R packages (via the Rcpp framework) to efficiently read BAM files using OpenMP-based multi-threading. SpliceWiz processes BAM files via the [processBAM](#) function, using a splicing and intron reference built from any given genome / gene annotation

resource using the [buildRef](#) function. [processBAM](#) generates two outputs per BAM file: a `txt.gz` file which is a gzip-compressed text file with multiple tables, containing information including junction read counts and intron retention metrics. This output is very similar to that of [IRFinder](#), as the analysis steps of SpliceWiz's BAM processing was built on an improved version of IRFinder's source code (version 1.3.1). Additionally, [processBAM](#) outputs a COV file, which is a binary bgzf-compressed file that contains strand-specific coverage data.

Once individual files have been analysed, SpliceWiz compiles a dataset using these individual outputs, using [collateData](#). This function unifies junctions detected across the dataset, and generates included / excluded counts of all putative IR events and annotated alternative splicing events (ASEs). This dataset is exported as a collection of files including an H5 database. The data is later imported into the R session using the [makeSE](#) function, as a [NxtSE](#) object.

The [NxtSE](#) object is a specialized [SummarizedExperiment](#) object tailored for use in SpliceWiz. Annotation of rows provide information about ASEs via [rowData](#), while columns allows users to provide annotations via [colData](#).

SpliceWiz offers several novel filters via the [ASEFilter](#) class. See [ASEFilter](#) for details.

Once the [NxtSE](#) is annotated and filtered, differential analysis is performed, using [limma](#), [Double-ExpSeq](#) (DES), [edgeR](#) and [DESeq2](#) wrappers. These wrappers model isoform counts as log-normal ([limma](#)), beta-binomial (DES) and negative-binomial ([edgeR](#) and [DESeq2](#)) distributions. See [ASE-methods](#) for details.

Finally, SpliceWiz provides visualisation tools to illustrate alternative splicing using coverage plots, including a novel method to normalise RNA-seq coverage grouped by experimental condition. This approach accounts for variations introduced by sequenced library size and gene expression. SpliceWiz efficiently computes and visualises means and variations in per-nucleotide coverage depth across alternate exons in genomic loci.

The main functions are:

- [Build-Reference-methods](#) - Prepares genome and gene annotation references from FASTA and GTF files and synthesizes the SpliceWiz reference for processing BAM files, collating the [NxtSE](#) object.
- [STAR-methods](#) - (Optional) Provides wrapper functions to build the STAR genome reference and alignment of short-read FASTQ raw sequencing files. This functionality is only available on systems with STAR installed.
- [processBAM](#) - OpenMP/C++ based algorithm to analyse single or multiple BAM files.
- [collateData](#) - Collates an experiment based on multiple IRFinder outputs for individual samples, into one unified H5-based data structure.
- [makeSE](#) - Constructs a [NxtSE](#) (H5-based [SummarizedExperiment](#)) object, specialised to house measurements of retained introns and junction counts of alternative splice events.
- [applyFilters](#) - Use default or custom filters to remove alternative splicing or IR events pertaining to low-abundance genes and transcripts.
- [ASE-methods](#) - one-step method to perform differential alternate splice event (ASE) analysis on a [NxtSE](#) object using [limma](#) or [DESeq2](#).
- [make_plot_data](#): Functions that compile individual and group-mean percent spliced in (PSI) values of IR and alternative splice events; useful to produce scatter plots or heatmaps.
- [Coverage](#): methods that retrieve coverage data from COV files.

- `getCoverageData` / `getPlotObject` / `plotView`: Functions for plotting SpliceWiz's novel coverage plots.

See the [SpliceWiz Quick-Start](#) for worked examples on how to use SpliceWiz [SpliceWiz Cookbook](#) for real-life usage examples

Author(s)

Alex Wong

References

Wong ACH, Wong JJ-L, Rasko JEJ, Schmitz U. SpliceWiz: interactive analysis and visualization of alternative splicing in R. Briefings in Bioinformatics, Volume 25, Issue 1, January 2024, bbad468. <https://doi.org/10.1093/bib/bbad468>

ASE-GLM-edgeR

Using Generalised linear models (GLMs) to analyse differential ASEs using edgeR

Description

These functions allow users to fit custom GLMs included/excluded counts using edgeR for differential Alternative Splice Events (ASEs)

Usage

```
fitASE_edgeR(  
  se,  
  strModelFormula,  
  strASEFormula,  
  useQL = TRUE,  
  IRmode = c("all", "annotated", "annotated_binary"),  
  filter_antiover = TRUE,  
  filter_antinear = FALSE  
)  
  
fitASE_edgeR_custom(  
  se,  
  model_IncExc,  
  model_ASE,  
  useQL = TRUE,  
  IRmode = c("all", "annotated", "annotated_binary"),  
  filter_antiover = TRUE,  
  filter_antinear = FALSE  
)  
  
testASE_edgeR(  
  se,  
  strModelFormula,  
  strASEFormula,  
  useQL = TRUE,  
  IRmode = c("all", "annotated", "annotated_binary"),  
  filter_antiover = TRUE,  
  filter_antinear = FALSE  
)
```

```

    se,
    fit,
    coef_IncExc = ncol(fit[["model_IncExc"]]),
    contrast_IncExc = NULL,
    coef_ASE = ncol(fit[["model_ASE"]]),
    contrast_ASE = NULL
)

addPSI_edgeR(results, se, condition, conditionList)

```

Arguments

<code>se</code>	The <code>NxtSE</code> object created by <code>makeSE()</code> . To reduce runtime and avoid excessive multiple testing, consider filtering the object using applyFilters
<code>strModelFormula</code>	A string specifying the model formula to fit isoform counts to assess differential expression in isolation. Should take the form of " <code>~0 + batch1 + batch2 + test_factor</code> ", where <code>batch1</code> and <code>batch2</code> are batch factors (if any), and <code>test_factor</code> is the variate of interest.
<code>strASEFormula</code>	A string specifying the model formula to fit PSIs (isoform ratios). The variate of interest should be specified as an interaction term with ASE. For example, following the above example, the ASE formula should be " <code>~0 + batch1 + batch2 + test_factor + test_factor:ASE</code> "
<code>useQL</code>	(default TRUE) Whether to use edgeR's quasi-likelihood method to help reduce false positives from near-zero junction / intron counts. NB: edgeR's quasi-likelihood method is run with legacy method (Lun and Smyth (2017)).
<code>IRmode</code>	(default all) Choose the approach to quantify IR events. Default <code>all</code> considers all introns as potentially retained, and calculates IR-ratio based on total splicing across the intron using the "SpliceOver" or "SpliceMax" approach (see collate-Data). Other options include <code>annotated</code> which calculates IR-ratios for annotated introns only, and <code>annotated_binary</code> which calculates PSI considering the "included" isoform as the IR-transcript, and the "excluded" transcript is quantified from splice counts only across the exact intron (but not that of overlapping introns). IR-ratio are denoted as "IR" events, whereas PSIs calculated using IR and intron-spliced binary alternatives are denoted as "RI" events.
<code>filter_antiover, filter_antinear</code>	Whether to remove novel IR events that overlap over or near anti-sense genes. Default will exclude antiover but not antinear introns. These are ignored if strand-specific RNA-seq protocols are used.
<code>model_IncExc</code>	A model matrix in which to model differential expression of isoform counts in isolation. The number of rows must equal that of the number of samples in <code>se</code>
<code>model_ASE</code>	A model matrix in which to model differential PSIs. The number of rows must be twice that of the number of samples in <code>se</code> , the first half are for included counts, and the second half are for excluded counts. See example below.
<code>fit</code>	The output returned by the <code>fitASE_edgeR</code> and <code>fitASE_edgeR_custom</code> functions.

<code>coef_IncExc</code> , <code>coef_ASE</code>	model coefficients to be dropped for LRT test between full and reduced models. Directly parsed onto <code>edgeR::glmQLFTest</code> . See <code>?edgeR::glmQLFTest</code> for details
<code>contrast_IncExc</code> , <code>contrast_ASE</code>	numeric vector specifying one or more #' contrasts of the linear model coefficients to be tested. Directly parsed onto <code>edgeR::glmQLFTest</code> . See <code>?edgeR::glmQLFTest</code> for details
<code>results</code>	The return value of <code>testASE_edgeR()</code> , to be used as input to append mean and delta PSI values onto.
<code>condition</code>	The name of the column containing the condition values in <code>colData(se)</code>
<code>conditionList</code>	A list (or vector) of condition values of which to calculate mean PSIs

Details

edgeR accounts appropriately for zero-counts which are often problematic as PSI approaches zero or one, leading to false positives. The following functions allow users to define model formulas to test relative expressions of included / excluded counts (to assess whether isoforms are differentially regulated, in isolation), as well as together as an interaction (the latter provides results of differential ASE analysis)

See the examples section for a brief explanation of how to use these functions.

See also [ASE-methods](#) for further explanations of results output.

Value

`fitASE_edgeR` and `fitASE_edgeR_custom` returns a named list containing the following:

- `IncExc` and `ASE` are DGEGLM objects containing the fitted models for isoform counts and PSIs, respectively
- `model_IncExc` and `model_ASE` are model matrices of the above fitted models.

`testASE_edgeR()` returns a `data.table` containing the following:

- `EventName`: The name of the ASE event. This identifies each ASE in downstream functions including [makeMeanPSI](#), [makeMatrix](#), and [plotCoverage](#)
- `EventType`: The type of event. See details section above.
- `EventRegion`: The genomic coordinates the event occupies. This spans the most upstream and most downstream splice junction involved in the ASE, and is use to guide the [plotCoverage](#) function.
- `flags`: Indicates which isoforms are NMD substrates and/or which are formed by novel splicing only.

edgeR specific output equivalent to statistics returned by `edgeR::topTags()`:

- `logFC`, `logCPM`, `F`, `PValue`, `FDR`: log fold change, log counts per million, F statistic, p value and (Benjamini Hochberg) adjusted p values of the differential PSIs for the contrasts or coefficients tested.

- `inc/exc(...)`: edgeR statistics corresponding to differential expression testing for raw included / excluded counts in isolation (not of the PSIs).

`addPSI_edgeR()` appends the following columns to the above output

- `AvgPSI_X`: the average percent spliced in / percent IR levels for condition X. Note this is a geometric mean, based on the arithmetic mean of logit PSI values.
- `deltaPSI`: The difference in PSI between the mean values of the two conditions.
- `abs_deltaPSI`: The absolute value of difference in PSI between the mean values of the two conditions.

Functions

- `fitASE_edgeR()`: Use edgeR to fit counts and ASE models with a given design formula
- `fitASE_edgeR_custom()`: Use edgeR to fit counts and ASE models with a given design formula
- `testASE_edgeR()`: Use edgeR to return differential ASE results. `coef` and `contrast` are parsed onto edgeR's `glmQLFTest` function
- `addPSI_edgeR()`: Adds average and delta PSIs of conditions of interest onto results produced by `testASE_edgeR()`. Note this is done automatically for other methods described in ASE-methods.

References

Lun A, Smyth G (2017). 'No counts, no variance: allowing for loss of degrees of freedom when assessing biological variability from RNA-seq data' *Stat Appl Genet Mol Biol*, 017 Apr 25;16(2):83-93. <https://doi.org/10.1515/sagmb-2017-0010>

Examples

```
# Load the NxtSE object and set up the annotations
# - see ?makeSE on example code of generating this NxtSE object
se <- SpliceWiz_example_NxtSE()

colData(se)$treatment <- rep(c("A", "B"), each = 3)
colData(se)$replicate <- rep(c("P", "Q", "R"), 2)
require("edgeR")

fit <- fitASE_edgeR(
  se,
  strModelFormula = "~0 + replicate + treatment",
  strASEFormula = "~0 + replicate + treatment + treatment:ASE"
)

# Get coefficient terms of Included / Excluded counts isolated model
colnames(fit$model_IncExc)
# [1] "replicateP" "replicateQ" "replicateR" "treatmentB"

# Get coefficient terms of PSI model
colnames(fit$model_ASE)
```



```

# [1] "replicateP" "replicateQ" "replicateR" "treatmentB"
# [5] "treatmentA:ASEIncluded" "treatmentB:ASEIncluded"

# Contrast between treatment "B" against treatment "A"
res <- testASE_edgeR(se, fit,
  contrast_IncExc = c(0,0,0,1),
  contrast_ASE = c(0,0,0,0,-1,1)
)

### # Add mean PSI values to results:
res_withPSI <- addPSI_edgeR(res, se, "treatment", c("B", "A"))

### Using custom model matrices to model counts
# - the equivalent analysis can be performed as follows:

# Sample annotations for isoform count expressions
colData <- as.data.frame(colData(se))

# Sample annotations for isoform count PSI analysis
colData_ASE <- rbind(colData, colData)
colData_ASE$ASE <- rep(c("Included", "Excluded"), each = nrow(colData))
rownames(colData_ASE) <- c(
  paste0(rownames(colData), ".Included"),
  paste0(rownames(colData), ".Excluded")
)

model_IncExc <- model.matrix(
  ~0 + replicate + treatment,
  data = colData
)

model_ASE <- model.matrix(
  ~0 + replicate + treatment + treatment:ASE,
  data = colData_ASE
)

fit <- fitASE_edgeR_custom(se, model_IncExc, model_ASE)

res_customModel <- testASE_edgeR(se, fit,
  contrast_IncExc = c(0,0,0,1),
  contrast_ASE = c(0,0,0,0,-1,1)
)

# Check this produces identical results:
identical(res_customModel, res)

### Time series examples using edgeR and splines
# - similar to section 4.8 in the edgeR vignette

colData(se)$timepoint <- rep(c(1,2,3), each = 2)
colData(se)$batch <- rep(c("1", "2"), 3)

```

```

# First, we set up a polynomial spline with 2 degrees of freedom:
Time <- poly(colData(se)$timepoint, df = 2)

# Next, we define the batch factor:
Batch <- factor(colData(se)$batch)

# Finally, we construct the same factors for ASE analysis. Note that
# each factor must be repeated twice

Time_ASE <- rbind(Time, Time)
Batch_ASE <- c(Batch, Batch)
ASE <- factor(
  rep(c("Included", "Excluded"), each = nrow(colData(se)))
)

# Now, we set up the model matrices for isoform and PSI count modelling
model_IncExc <- model.matrix(~0 + Batch + Time)
model_ASE <- model.matrix(~0 + Batch_ASE + Time_ASE + Time_ASE:ASE)

fit <- fitASE_edgeR_custom(se, model_IncExc, model_ASE)

# Note the coefficients of interest in the constructed models:

colnames(model_IncExc)
# [1] "Batch1" "Batch2" "Time1" "Time2"

colnames(model_ASE)
# [1] "Batch_ASE1" "Batch_ASE2" "Time_ASE1" "Time_ASE2"
# [5] "Time_ASE1:ASEIncluded" "Time_ASE2:ASEIncluded"

# We are interested in a model in which `Time` is excluded, thus:

res <- testASE_edgeR(se, fit,
  coef_IncExc = 3:4,
  coef_ASE = 5:6
)

# Finally, add PSI values for each time point:

res_withPSI <- addPSI_edgeR(res, se, "timepoint", c(1, 2, 3))

```

Description

Use Limma, DESeq2, DoubleExpSeq, and edgeR wrapper functions to test for differential Alternative Splice Events (ASEs)

Usage

```
ASE_limma(  
  se,  
  test_factor,  
  test_nom,  
  test_denom,  
  batch1 = "",  
  batch2 = "",  
  IRmode = c("all", "annotated", "annotated_binary"),  
  filter_antiover = TRUE,  
  filter_antinear = FALSE  
)
```

```
ASE_edgeR(  
  se,  
  test_factor,  
  test_nom,  
  test_denom,  
  batch1 = "",  
  batch2 = "",  
  useQL = TRUE,  
  IRmode = c("all", "annotated", "annotated_binary"),  
  filter_antiover = TRUE,  
  filter_antinear = FALSE  
)
```

```
ASE_limma_timeseries(  
  se,  
  test_factor,  
  batch1 = "",  
  batch2 = "",  
  degrees_of_freedom = 1,  
  IRmode = c("all", "annotated", "annotated_binary"),  
  filter_antiover = TRUE,  
  filter_antinear = FALSE  
)
```

```
ASE_edgeR_timeseries(  
  se,  
  test_factor,  
  batch1 = "",  
  batch2 = "",  
  degrees_of_freedom = 1,  
  useQL = TRUE,  
  IRmode = c("all", "annotated", "annotated_binary"),  
  filter_antiover = TRUE,  
  filter_antinear = FALSE  
)
```

```

ASE_DESeq(
  se,
  test_factor,
  test_nom,
  test_denom,
  batch1 = "",
  batch2 = "",
  n_threads = 1,
  IRmode = c("all", "annotated", "annotated_binary"),
  filter_antiover = TRUE,
  filter_antinear = FALSE
)

ASE_DoubleExpSeq(
  se,
  test_factor,
  test_nom,
  test_denom,
  IRmode = c("all", "annotated", "annotated_binary"),
  filter_antiover = TRUE,
  filter_antinear = FALSE
)

```

Arguments

se	The <code>NxtSE</code> object created by <code>makeSE()</code> . To reduce runtime and avoid excessive multiple testing, consider filtering the object using applyFilters
test_factor	The column name in the sample annotation <code>colData(se)</code> that contains the desired variables to be contrasted. For <code>ASE_limma_timeseries()</code> and <code>ASE_DESeq()</code> (when <code>test_nom</code> and <code>test_denom</code> parameters are left blank), <code>test_factor</code> must contain numerical values representing the time variable.
test_nom	The nominator condition to test for differential ASE. Usually the "treatment" condition
test_denom	The denominator condition to test against for differential ASE. Usually the "control" condition
batch1, batch2	(Optional, <code>limma</code> and <code>DESeq2</code> only) One or two condition types containing batch information to account for.
IRmode	(default <code>all</code>) Choose the approach to quantify IR events. Default <code>all</code> considers all introns as potentially retained, and calculates IR-ratio based on total splicing across the intron using the "SpliceOver" or "SpliceMax" approach (see collate-Data). Other options include <code>annotated</code> which calculates IR-ratios for annotated introns only, and <code>annotated_binary</code> which calculates PSI considering the "included" isoform as the IR-transcript, and the "excluded" transcript is quantified from splice counts only across the exact intron (but not that of overlapping introns). IR-ratio are denoted as "IR" events, whereas PSIs calculated using IR and intron-spliced binary alternatives are denoted as "RI" events.

<code>filter_antiover, filter_antinear</code>	Whether to remove novel IR events that overlap over or near anti-sense genes. Default will exclude antiover but not antinear introns. These are ignored if strand-specific RNA-seq protocols are used.
<code>useQL</code>	(default TRUE) Whether to use edgeR's quasi-likelihood method to help reduce false positives from near-zero junction / intron counts. NB: edgeR's quasi-likelihood method is run with legacy method (Lun and Smyth (2017)).
<code>degrees_of_freedom</code>	(default 1) The complexity of time series trends modeled by <code>ASE_limma_timeseries</code> and <code>ASE_edgeR_timeseries</code> . E.g., 1 will only model linear trends, 2 extends the capacity for quadratic trends, and 3 for cubic trends, etc.
<code>n_threads</code>	(DESeq2 only) How many threads to use for DESeq2 based analysis.

Details

Using **limma**, SpliceWiz models included and excluded counts as log-normal distributed, whereas using **DESeq2**, SpliceWiz models included and excluded counts as negative binomial distributed with dispersion shrinkage according to their mean count expressions. For **limma** and **DESeq2**, differential ASE are considered as the "interaction" between included and excluded splice counts for each sample. See [this vignette](#) for an explanation of how this is done.

SpliceWiz's **limma** wrapper implements an additional filter where ASEs with an average cpm values of either Included or Excluded counts are less than 1. **DESeq2** has its own method for handling outliers, which seems to work well for handling situations where $PSI \sim 0$ or $PSI \sim 1$.

Time series are supported by SpliceWiz to a limited extent. Time series analysis can be performed via **limma** or **DESeq2**. For **limma** time-series analysis, use `ASE_limma_timeseries()`, specifying the `test_factor` as the column of numeric values containing time series data. For **DESeq2**, time series differential analysis can be activated using the `ASE_DESeq()` function, again specifying `test_factor` as the column containing time series data (and leaving `test_nom` and `test_denom` parameters blank). See examples below.

edgeR models counts using a negative binomial model. It accounts appropriately for zero-counts which are often problematic as PSI approaches zero or one, leading to false positives. The **edgeR**-based option produces differential ASEs that are less biased towards low counts. Our preliminary analysis shows it to be more accurate than **limma** or **DoubleExpSeq** based methods.

For time series analysis using **edgeR**, `ASE_edgeR_timeseries()` can be used interchangeably with its counterpart **limma**-based function. For complex models, please see [ASE-GLM-edgeR](#) to build your own GLM models.

Using **DoubleExpSeq**, included and excluded counts are modeled using the generalized beta prime distribution, using empirical Bayes shrinkage to estimate dispersion.

EventType are as follow:

- IR = intron retention (IR-ratio) - all introns are considered
- MXE = mutually exclusive exons
- SE = skipped exons
- AFE = alternate first exon
- ALE = alternate last exon

- A5SS = alternate 5'-splice site
- A3SS = alternate 3'-splice site
- RI = (known / annotated) intron retention (PSI).

NB: SpliceWiz measures intron retention events using two different approaches, the choice of which is left to the user - see [ASE-methods](#):

- **IR** (intron retention) events: considers all introns to be potentially retained. Given in most scenarios there may be uncertainty as to which of the many mutually-overlapping introns are spliced to produce the major isoform, SpliceWiz adopts the IRFinder approach by using the IR-ratio. The "included" isoform is the relative abundance of the IR-transcript, as approximated by the trimmed-mean depth of coverage across the intron (excluding outliers including exons of other transcripts, intronic elements such as snoRNAs, etc). The "excluded isoform" includes **all** spliced transcripts that contain an overlapping intron, as estimated via SpliceWiz's SpliceOver and IRFinder's SpliceMax methods - see [collateData](#).
- **RI** (annotated retained introns) considers only annotated retained introns, i.e., those annotated within the given reference. These are quantified using PSI, considering the included (IR-transcript) and excluded (splicing of the exact intron) as binary alternatives.

SpliceWiz considers "included" counts as those that represent abundance of the "included" isoform, whereas "excluded" counts represent the abundance of the "excluded" isoform. To allow comparison between modalities, SpliceWiz applies a convention whereby the "included" transcript is one where its splice junctions are by definition shorter than those of "excluded" transcripts. Specifically, this means the included / excluded isoforms are as follows:

EventType	Included	Excluded
IR or RI	Intron Retention	Spliced Intron
MXE	Upstream exon inclusion	Downstream exon inclusion
SE	Exon inclusion	Exon skipping
AFE	Downstream exon usage	Upstream exon usage
ALE	Upstream exon usage	Downstream exon usage
A5SS	Downstream 5'-SS	Upstream 5'-SS
A3SS	Upstream 3'-SS	Downstream 3'-SS

Value

For all methods, a `data.table` containing the following:

- **EventName**: The name of the ASE event. This identifies each ASE in downstream functions including [makeMeanPSI](#), [makeMatrix](#), and [plotCoverage](#)
- **EventType**: The type of event. See details section above.
- **EventRegion**: The genomic coordinates the event occupies. This spans the most upstream and most downstream splice junction involved in the ASE, and is used to guide the [plotCoverage](#) function.
- **flags**: Indicates which isoforms are NMD substrates and/or which are formed by novel splicing only.

- AvgPSI_nom, Avg_PSI_denom: the average percent spliced in / percent IR levels for the two conditions being contrasted. nom and denom in column names are replaced with the condition names. Note this is a geometric mean, based on the arithmetic mean of logit PSI values.
- deltaPSI: The difference in PSI between the mean values of the two conditions.
- abs_deltaPSI: The absolute value of difference in PSI between the mean values of the two conditions.

limma specific output

- logFC, AveExpr, t, P.Value, adj.P.Val, B: limma topTable columns of differential ASE. See [limma::topTable](#) for details.
- inc/exc_(logFC, AveExpr, t, P.Value, adj.P.Val, B): limma results for differential testing for raw included / excluded counts only

edgeR specific output equivalent to statistics returned by [edgeR::topTags](#):

- logFC, logCPM, F, PValue, FDR: log fold change, log counts per million, F statistic, p value and (Benjamini Hochberg) adjusted p values.
- inc/exc_(...): edgeR statistics corresponding to differential expression testing for raw included / excluded counts in isolation

DESeq2 specific output

- baseMean, log2FoldChange, lfcSE, stat, pvalue, padj: DESeq2 results columns for differential ASE; see [DESeq2::results](#) for details.
- inc/exc_(baseMean, log2FoldChange, lfcSE, stat, pvalue, padj): DESeq2 results for differential testing for raw included / excluded counts only

DoubleExp specific output

- MLE_nom, MLE_denom: Maximum likelihood expectation of PSI values for the denom in column names are replaced with the condition names
- MLE_LFC: Log2-fold change of the MLE
- P.Value, adj.P.Val: Nominal and BH-adjusted P values
- n_eff: Number of effective samples (i.e. non-zero or non-unity PSI)
- mDepth: Mean Depth of splice coverage in each of the two groups.
- Dispersion_Reduced, Dispersion_Full: Dispersion values for reduced and full models. See [DoubleExpSeq::DBGLM1](#) for details.

Functions

- ASE_limma(): Use limma to perform differential ASE analysis of a filtered NxtSE object
- ASE_edgeR(): Use edgeR to perform differential ASE analysis of a filtered NxtSE object
- ASE_limma_timeseries(): Use limma to perform differential ASE analysis of a filtered NxtSE object (time series)
- ASE_edgeR_timeseries(): Use edgeR to perform differential time series of a filtered NxtSE object

- ASE_DESeq(): Use DESeq2 to perform differential ASE analysis of a filtered NxtSE object
- ASE_DoubleExpSeq(): Use DoubleExpSeq to perform differential ASE analysis of a filtered NxtSE object (uses double exponential beta-binomial model) to estimate group dispersions, followed by LRT

References

Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK (2015). 'limma powers differential expression analyses for RNA-sequencing and microarray studies.' *Nucleic Acids Research*, 43(7), e47. <https://doi.org/10.1093/nar/gkv007>

Love MI, Huber W, Anders S (2014). 'Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2.' *Genome Biology*, 15, 550. <https://doi.org/10.1186/s13059-014-0550-8>

Ruddy S, Johnson M, Purdom E (2016). 'Shrinkage of dispersion parameters in the binomial family, with application to differential exon skipping.' *Ann. Appl. Stat.* 10(2): 690-725. <https://doi.org/10.1214/15-A0AS871>

Gilis J, Vitting-Seerup K, Van den Berge K, Clement L (2021). 'Scalable analysis of differential transcript usage for bulk and single-cell RNA-sequencing applications.' *F1000Research* 2021, 10:374. <https://doi.org/10.12688/f1000research.51749.1>

Lun A, Smyth G (2017). 'No counts, no variance: allowing for loss of degrees of freedom when assessing biological variability from RNA-seq data' *Stat Appl Genet Mol Biol*, 017 Apr 25;16(2):83-93. <https://doi.org/10.1515/sagmb-2017-0010>

Examples

```
# Load the NxtSE object and set up the annotations
# - see ?makeSE on example code of generating this NxtSE object
se <- SpliceWiz_example_NxtSE(novelSplicing = TRUE)

colData(se)$treatment <- rep(c("A", "B"), each = 3)
colData(se)$replicate <- rep(c("P","Q","R"), 2)

# Limma analysis (counts modeled using log-normal distribution)

require("limma")
res_limma <- ASE_limma(se, "treatment", "A", "B")

# edgeR analysis (counts modeled using negative binomial distribution)
# - QL: whether quasi-likelihood method was used

require("edgeR")
res_edgeR <- ASE_edgeR(se, "treatment", "A", "B", useQL = FALSE)
res_edgeR_QL <- ASE_edgeR(se, "treatment", "A", "B", useQL = TRUE)

# DoubleExpSeq analysis (counts modeled using beta binomial distribution)

require("DoubleExpSeq")
res_DES <- ASE_DoubleExpSeq(se, "treatment", "A", "B")

# DESeq2 analysis (counts modeled using negative binomial distribution)
```



```

require("DESeq2")
res_DESeq <- ASE_DESeq(se, "treatment", "A", "B")

# Time series examples

colData(se)$timepoint <- rep(c(1,2,3), each = 2)
colData(se)$batch <- rep(c("1", "2"), 3)

res_limma_timeseries <- ASE_limma_timeseries(se, "timepoint")
res_edgeR_timeseries <- ASE_edgeR_timeseries(se, "timepoint")
res_DESeq_timeseries <- ASE_DESeq(se, "timepoint")

```

ASEFilter-class	<i>SpliceWiz filters to remove low-confidence alternative splicing and intron retention events</i>
-----------------	--

Description

SpliceWiz implements a number of novel filters designed to exclude alternative splicing events (ASEs) that yield low-confidence estimates.

Usage

```

ASEFilter(
  filterClass = c("Data", "Annotation"),
  filterType = c("Depth", "Participation", "Consistency", "Modality", "Protein_Coding",
    "NMD", "TSL", "Terminus", "ExclusiveMXE", "StrictAltSS"),
  pcTRUE = 100,
  minimum = 20,
  maximum = 1,
  minDepth = 5,
  condition = "",
  minCond = -1,
  EventTypes = c("IR", "MXE", "SE", "A3SS", "A5SS", "AFE", "ALE", "RI")
)

```

Arguments

filterClass	Must be either "Data" or "Annotation". See details
filterType	Must be a valid "Data" or "Annotation" filter. See details
pcTRUE	If conditions are set, what percentage of all samples in each of the condition must satisfy the filter for the event to pass the filter check. Must be between 0 and 100 (default 100)
minimum	Filter-dependent argument. See details
maximum	Filter-dependent argument. See details
minDepth	Filter-dependent argument. See details

condition	(default "") If set, must match the name of an experimental condition in the NxtSE object to be filtered, i.e. a column name in colData(se). Leave blank to disable filtering by condition
minCond	(default -1) If condition is set, how many minimum number of conditions must pass the filter criteria. For example, if condition = "Batch", and batches are "A", "B", or "C", setting minCond = 2 with pctTRUE = 100 means that all samples belonging to two of the three types of Batch must pass the filter criteria. Setting -1 means all elements of condition must pass criteria. Set to -1 when the number of elements in the experimental condition is unknown. Ignored if condition is left blank.
EventTypes	What types of events are considered for filtering. Must be one or more of c("IR", "MXE", "SE", "A3SS", "A5SS", "AFE", "ALE", "RI"). Events not specified in EventTypes are not filtered (i.e. they will pass the filter without checks)

Details

Annotation Filters

- **Modality:** Filters for specific modalities of ASEs. All events belonging to the specified EventTypes are removed. No additional parameters required.
- **Protein_Coding:** Filters for alternative splicing or IR events involving protein-coding transcripts. No additional parameters required.
- **NMD:** Filters for events in which one isoform is a predicted NMD substrate.
- **TSL:** filters for events in which both isoforms have a TSL level below or equal to minimum
- **Terminus:** In alternate first exons, the splice junction must not be shared with another transcript for which it is not its first intron. For alternative last exons, the splice junction must not be shared with another transcript for which it is not its last intron
- **ExclusiveMXE:** For MXE events, the two alternate cassette exons must not overlap in their genomic regions
- **StrictAltSS:** For A5SS / A3SS events, the two alternate splice sites must not be interrupted by detected introns

Data Filters

- **Depth:** Filters IR or alternative splicing events of transcripts that are "expressed" with adequate Depth as calculated by the sum of all splicing and IR reads spanning the event. Events with Depth below minimum are filtered out
- **Participation:** Participation means different things to IR and alternative splicing.

For **IR**, Participation refers to the percentage of the measured intron covered with reads. Only introns of samples with a depth of intron coverage (intron depth) above minDepth are assessed, where introns with coverage percentage below minimum are filtered out.

For **non-IR ASEs**, Participation refers to the percentage of all splicing events observed across the genomic region (SpliceOver metric) that is compatible with either the included or excluded event. This prevents SpliceWiz from doing differential analysis between two minor isoforms.

Instead of IntronDepth, in AS events SpliceWiz considers events where the SpliceOver metric exceed minDepth. Then, events with a SpliceOver metric below minimum are excluded.

We recommend testing IR events for > 70% coverage and AS events for > 40% coverage as given in the default filters which can be accessed using [getDefaultFilters](#)

- **Consistency:** Skipped exons (SE) and mutually exclusive exons (MXE) comprise reads aligned to two contiguous splice junctions. Most algorithms take the average counts from both junctions. This will inadvertently include transcripts that share one but not both splice events. To check that this is not happening, we require both splice junctions to have comparable counts. This filter checks whether reads from each splice junction comprises a reasonable proportion of the sum of these reads.

Events are excluded if either of the upstream or downstream event is lower than total splicing events by a log-2 magnitude above maximum. For example, if maximum = 2, we require both upstream and downstream events to represent at least $1/(2^2) = 1/4$ of the sum of upstream and downstream event. If maximum = 3, then each junction must be at least 1/8 of total, etc. This is considered for each isoform of each event, and is NOT tested when total (upstream+downstream) counts belonging to each isoform is below minDepth.

IR-events are also checked. For IR events, the upstream and downstream exon-intron spanning reads must comprise a reasonable proportion of total exon-intron spanning reads.

We highly recommend using the default filters, which can be acquired using [getDefaultFilters](#)

Value

An ASEFilter object with the specified parameters

Functions

- `ASEFilter()`: Constructs a ASEFilter object

See Also

[Run_SpliceWiz_Filters](#)

Examples

```
# Create a ASEFilter that filters for protein-coding ASE
f1 <- ASEFilter(filterClass = "Annotation", filterType = "Protein_Coding")

# Create a ASEFilter that filters for Depth >= 20 in IR events
f2 <- ASEFilter(
  filterClass = "Data", filterType = "Depth",
  minimum = 20, EventTypes = c("IR", "RI")
)

# Create a ASEFilter that filters for Participation > 60% in splice events
```

```

# that must be satisfied in at least 2 categories of condition "Genotype"
f3 <- ASEFilter(
  filterClass = "Data", filterType = "Participation",
  minimum = 60, EventTypes = c("MXE", "SE", "AFE", "ALE", "A3SS", "A5SS"),
  condition = "Genotype", minCond = 2
)

# Create a ASEFilter that filters for Depth > 10 in all events
# that must be satisfied in at least 50% of each gender
f4 <- ASEFilter(
  filterClass = "Data", filterType = "Depth",
  minimum = 10, condition = "gender", pcTRUE = 50
)

# Get a description of what these filters do:
f1
f2
f3
f4

```

Build-Reference-methods

Builds reference files used by SpliceWiz

Description

This function builds the reference required by the SpliceWiz engine, as well as alternative splicing annotation data for SpliceWiz. See examples below for guides to making the SpliceWiz reference.

Usage

```

getResources(
  reference_path = "./Reference",
  fasta = "",
  gtf = "",
  overwrite = FALSE,
  force_download = FALSE,
  verbose = TRUE
)

buildRef(
  reference_path = "./Reference",
  fasta = "",
  gtf = "",
  overwrite = FALSE,
  force_download = FALSE,
  chromosome_aliases = NULL,

```

```

    genome_type = "",
    nonPolyARef = "",
    MappabilityRef = "",
    BlacklistRef = "",
    ontologySpecies = "",
    useExtendedTranscripts = TRUE,
    lowMemoryMode = TRUE,
    verbose = TRUE
)

buildFullRef(
  reference_path = "./Reference",
  fasta = "",
  gtf = "",
  use_STAR_mappability = FALSE,
  overwrite = FALSE,
  force_download = FALSE,
  chromosome_aliases = NULL,
  genome_type = "",
  nonPolyARef = "",
  MappabilityRef = "",
  BlacklistRef = "",
  ontologySpecies = "",
  useExtendedTranscripts = TRUE,
  verbose = TRUE,
  n_threads = 4,
  ...
)

getNonPolyARef(genome_type)

getAvailableGO(localHub = FALSE, ah = AnnotationHub(localHub = localHub))

```

Arguments

<code>reference_path</code>	(REQUIRED) The directory path to store the generated reference files
<code>fasta</code>	The file path or web link to the user-supplied genome FASTA file. Alternatively, the name of the AnnotationHub record containing the genome resource. May be omitted if <code>getResources()</code> has already been run using the same <code>reference_path</code> .
<code>gtf</code>	The file path or web link to the user-supplied transcript GTF file (or gzipped GTF file). Alternatively, the name of the AnnotationHub record containing the transcript GTF file. May be omitted if <code>getResources()</code> has already been run using the same <code>reference_path</code> .
<code>overwrite</code>	(default FALSE) For <code>getResources()</code> : if the genome FASTA and gene annotation GTF files already exist in the resource subdirectory, it will not be overwritten. For <code>buildRef()</code> and <code>buildFullRef()</code> : the SpliceWiz reference will not be overwritten if one already exist. A reference is considered to exist if the file <code>SpliceWiz.ref.gz</code> is present inside <code>reference_path</code> .

- `force_download` (default FALSE) When online resources are retrieved, a local copy is stored in the SpliceWiz BiocFileCache. Subsequent calls to the web resource will fetch the local copy. Set `force_download` to TRUE will force the resource to be downloaded from the web. Set this to TRUE only if the web resource has been updated since the last retrieval.
- `verbose` (default TRUE) If FALSE, will silence progress messages
- `chromosome_aliases` (Highly optional) A 2-column data frame containing chromosome name conversions. If this is set, allows `processBAM` to parse BAM alignments to a genome whose chromosomes are named differently to the reference genome. The most common scenario is where Ensembl genome typically use chromosomes "1", "2", ..., "X", "Y", whereas UCSC/Gencode genome use "chr1", "chr2", ..., "chrX", "chrY". See example below. Refer to <https://github.com/dpryan79/ChromosomeMappings> for a list of chromosome alias resources.
- `genome_type` Allows `buildRef()` to select default `nonPolyARef` and `MappabilityRef` for selected genomes. Allowed options are: hg38, hg19, mm10, and mm9.
- `nonPolyARef` (Optional) A BED file of regions defining known non-polyadenylated transcripts. This file is used for QC analysis to measure Poly-A enrichment quality of samples. An RDS file (openable using `readRDS()`) of a `GRanges` object is acceptable. If omitted, and `genome_type` is defined, the default for the specified genome will be used.
- `MappabilityRef` (Optional) A BED file of low mappability regions due to repeat elements in the genome. If omitted, the file generated by `calculateMappability()` will be used where available, and if this is not, the default file for the specified `genome_type` will be used. If `genome_type` is not specified, `MappabilityRef` is not used. An RDS file (openable using `readRDS()`) of a `GRanges` object is acceptable. See details.
- `BlacklistRef` A BED file of regions to be otherwise excluded from IR analysis. If omitted, a blacklist is not used (this is the default). An RDS file (openable using `readRDS()`) of a `GRanges` object is acceptable.
- `ontologySpecies` (default "") The species for which gene ontology classifications should be fetched from AnnotationHub. Ignored if `genome_type` is set (as human or mouse GO will be used instead).
- `useExtendedTranscripts` (default TRUE) Should non-protein-coding transcripts such as anti-sense and lincRNA transcripts be included in searching for IR / AS events? Setting FALSE (vanilla IRFinder) will exclude transcripts other than `protein_coding` and `processed_transcript` transcripts from IR analysis.
- `lowMemoryMode` (default TRUE) By default, SpliceWiz converts FASTA files to TwoBit, then uses the TwoBit file to fetch genome sequences. In most cases, this method uses less memory and is faster, but can be very slow on some systems. Set this option to FALSE (which will convert the TwoBit file back to FASTA) if you experience very slow genome fetching (e.g. when annotating splice motifs).
- `use_STAR_mappability` (default FALSE) In `buildFullRef()`, whether to run `STAR_mappability` to calculate low-mappability regions. We recommend setting this to FALSE for the

	common genomes (human and mouse), and to TRUE for genomes not supported by <code>genome_type</code> . When set to false, the <code>MappabilityExclusion</code> default file corresponding to <code>genome_type</code> will automatically be used.
<code>n_threads</code>	The number of threads used to generate the STAR reference and mappability calculations. Multi-threading is not used for SpliceWiz reference generation (but multiple cores are utilised in data-table and fst file processing automatically, where available). See STAR-methods
<code>...</code>	For <code>buildFullRef()</code> , additional parameters to be parsed into <code>STAR_buildRef</code> which <code>buildFullRef()</code> runs internally. See STAR_buildRef
<code>localHub</code>	(default FALSE) For <code>getAvailableGO()</code> , whether to use offline mode for AnnotationHub resources. If TRUE, offline mode will be used.
<code>ah</code>	For <code>getAvailableGO()</code> , the AnnotationHub object. Leave as default to use the entirety of AnnotationHub resources.

Details

`getResources()` processes the files, downloads resources from web links or from `AnnotationHub()`, and saves a local copy in the "resource" subdirectory within the given `reference_path`. Resources are retrieved via either:

1. User-supplied FASTA and GTF file. This can be a file path, or a web link (e.g. `'http://'`, `'https://'` or `'ftp://'`). Use `fasta` and `gtf` to specify the files or web paths to use.
2. AnnotationHub genome and gene annotation (Ensembl): supply the names of the genome sequence and gene annotations to `fasta` and `gtf`.

`buildRef()` will first run `getResources()` if resources are not yet saved locally (i.e. `getResources()` is not already run). Then, it creates the SpliceWiz references. Typical run-times are 5 to 10 minutes for human and mouse genomes (after resources are downloaded).

NB: the parameters `fasta` and `gtf` can be omitted in `buildRef()` if `getResources()` is already run.

`buildFullRef()` builds the STAR aligner reference alongside the SpliceWiz reference. The STAR reference will be located in the STAR subdirectory of the specified reference path. If `use_STAR_mappability` is set to TRUE this function will empirically compute regions of low mappability. This function requires STAR to be installed on the system (which only runs on linux-based systems).

`getNonPolyARef()` returns the path of the non-polyA reference file for the human and mouse genomes.

Typical usage involves running `buildRef()` for human and mouse genomes and specifying the `genome_type` to use the default `MappabilityRef` and `nonPolyARef` files for the specified genome. For non-human non-mouse genomes, use one of the following alternatives:

- Create the SpliceWiz reference without using Mappability Exclusion regions. To do this, simply run `buildRef()` and omit `MappabilityRef`. This is acceptable assuming the introns assessed are short and do not contain intronic repeats
- Calculating Mappability Exclusion regions using the STAR aligner, and building the SpliceWiz reference. This can be done using the `buildFullRef()` function, on systems where STAR is installed

- Instead of using the STAR aligner, any genome splice-aware aligner could be used. See [Mappability-methods](#) for an example workflow using the Rsubread aligner. After producing the `MappabilityExclusion.bed.gz` file (in the `Mappability` subfolder), run `buildRef()` using this file (or simply leave it blank).

BED files are tab-separated text files containing 3 unnamed columns specifying chromosome, start and end coordinates. To view an example BED file, open the file specified in the path returned by `getNonPolyARef("hg38")`

If `MappabilityRef`, `nonPolyARef` and `BlacklistRef` are left blank, the following will be used (by priority):

1. The previously used `Mappability`, `non-polyA` and/or `Blacklist` file resource from a previous run, if available,
2. The resource implied by the `genome_type` parameter, if specified,
3. No resource is used.

To rebuild a SpliceWiz reference using existing resources This is typically run when updating an old resource to a new SpliceWiz version. Simply run `buildRef()`, specifying the existing reference directory, leave the `fasta` and `gtf` parameters blank, and set `overwrite = TRUE`. SpliceWiz will use the previously-used resources to re-create the reference.

See examples below for common use cases.

Value

For `getResources`: creates the following local resources:

- `reference_path/resource/genome.2bit`: Local copy of the genome sequences as a `TwoBit-File`.
- `reference_path/resource/transcripts.gtf.gz`: Local copy of the gene annotation as a gzip-compressed file.

For `buildRef()` and `buildFullRef()`: creates a SpliceWiz reference which is written to the given directory specified by `reference_path`. Files created includes:

- `reference_path/settings.Rds`: An RDS file containing parameters used to generate the SpliceWiz reference
- `reference_path/SpliceWiz.ref.gz`: A gzipped text file containing collated SpliceWiz reference files. This file is used by [processBAM](#)
- `reference_path/fst/`: Contains `fst` files for subsequent easy access to SpliceWiz generated references
- `reference_path/cov_data.Rds`: An RDS file containing data required to visualise genome / transcript tracks.

`buildFullRef()` also creates a STAR reference located in the `STAR` subdirectory inside the designated `reference_path`

For `getNonPolyARef()`: Returns the file path to the BED file for the nonPolyA loci for the specified genome.

For `getAvailableGO()`: Returns a vector containing names of species with supported gene ontology annotations.

Functions

- `getResources()`: Processes / downloads a copy of the genome and gene annotations and stores this in the "resource" subdirectory of the given reference path
- `buildRef()`: First calls `getResources()` (if required). Afterwards creates the SpliceWiz reference in the given reference path
- `buildFullRef()`: One-step function that fetches resources, creates a STAR reference (including mappability calculations), then creates the SpliceWiz reference
- `getNonPolyARef()`: Returns the path to the BED file containing coordinates of known non-polyadenylated transcripts for genomes hg38, hg19, mm10 and mm9,
- `getAvailableG0()`: Returns available species on Bioconductor's AnnotationHub. Currently, only Bioconductor's OrgDb/Ensembl gene ontology annotations are supported.

See Also

[Mappability-methods](#) for methods to calculate low mappability regions

[STAR-methods](#) for a list of STAR wrapper functions

[AnnotationHub](#)

<https://github.com/alexchwong/SpliceWizResources> for RDS files of Mappability Exclusion GRanges objects (for hg38, hg19, mm10 and mm9) that can be use as input files for `MappabilityRef` in `buildRef()`. These resources are intended for SpliceWiz users on older Bioconductor versions (3.13 or earlier)

Examples

```
# Quick runnable example: generate a reference using SpliceWiz's example genome
```

```
example_ref <- file.path(tempdir(), "Reference")
getResources(
  reference_path = example_ref,
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)
buildRef(
  reference_path = example_ref
)
```

```
# NB: the above is equivalent to:
```

```
example_ref <- file.path(tempdir(), "Reference")
buildRef(
  reference_path = example_ref,
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)
```

```
# Get the path to the Non-PolyA BED file for hg19
```

```

getNonPolyARef("hg19")

# View available species for AnnotationHub's Ensembl/orgDB-based GO resources

availSpecies <- getAvailableGO()

# Build example reference with `Homo sapiens` Ens/orgDB gene ontology

ont_ref <- file.path(tempdir(), "Reference_withGO")
buildRef(
  reference_path = ont_ref,
  fasta = chrZ_genome(),
  gtf = chrZ_gtf(),
  ontologySpecies = "Homo sapiens"
)

## Not run:

### Long examples ###

# Generate a SpliceWiz reference from user supplied FASTA and GTF files for a
# hg38-based genome:

buildRef(
  reference_path = "./Reference_user",
  fasta = "genome.fa", gtf = "transcripts.gtf",
  genome_type = "hg38"
)

# NB: Setting `genome_type = hg38`, will automatically use default
# nonPolyARef and MappabilityRef for `hg38`

# Reference generation from Ensembl's FTP links:

FTP <- "ftp://ftp.ensembl.org/pub/release-94/"
buildRef(
  reference_path = "./Reference_FTP",
  fasta = paste0(FTP, "fasta/homo_sapiens/dna/",
    "Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz"),
  gtf = paste0(FTP, "gtf/homo_sapiens/",
    "Homo_sapiens.GRCh38.94.chr.gtf.gz"),
  genome_type = "hg38"
)

# Get AnnotationHub record names for Ensembl release-94:

# First, search for the relevant AnnotationHub record names:

ah <- AnnotationHub::AnnotationHub()
AnnotationHub::query(ah, c("Homo Sapiens", "release-94"))

buildRef(

```

```
    reference_path = "./Reference_AH",
    fasta = "AH65745",
    gtf = "AH64631",
    genome_type = "hg38"
)

# Build a SpliceWiz reference, setting chromosome aliases to allow
# this reference to process BAM files aligned to UCSC-style genomes:

chrom.df <- GenomeInfoDb::genomeStyles()$Homo_sapiens

buildRef(
  reference_path = "./Reference_UCSC",
  fasta = "AH65745",
  gtf = "AH64631",
  genome_type = "hg38",
  chromosome_aliases = chrom.df[, c("Ensembl", "UCSC")]
)

# One-step generation of SpliceWiz and STAR references, using 4 threads.
# NB1: requires a linux-based system with STAR installed.
# NB2: A STAR reference genome will be generated in the `STAR` subfolder
#       inside the given `reference_path`.
# NB3: A custom Mappability Exclusion file will be calculated using STAR
#       and will be used to generate the SpliceWiz reference.

buildFullRef(
  reference_path = "./Reference_with_STAR",
  fasta = "genome.fa", gtf = "transcripts.gtf",
  genome_type = "hg38",
  use_STAR_mappability = TRUE,
  n_threads = 4
)

# NB: the above is equivalent to running the following in sequence:

getResources(
  reference_path = "./Reference_with_STAR",
  fasta = "genome.fa", gtf = "transcripts.gtf"
)
STAR_buildRef(
  reference_path = reference_path,
  also_generate_mappability = TRUE,
  n_threads = 4
)
buildRef(
  reference_path = "./Reference_with_STAR",
  genome_type = ""
)

## End(Not run)
```

collateData	<i>Collates a dataset from (processBAM) output files of individual samples</i>
-------------	--

Description

collateData() creates a dataset from a collection of [processBAM](#) output files belonging to an experiment.

Usage

```
collateData(
  Experiment,
  reference_path,
  output_path,
  IRMode = c("SpliceOver", "SpliceMax"),
  packageCOVfiles = FALSE,
  novelSplicing = FALSE,
  forceStrandAgnostic = FALSE,
  novelSplicing_minSamples = 3,
  novelSplicing_countThreshold = 10,
  novelSplicing_minSamplesAboveThreshold = 1,
  novelSplicing_requireOneAnnotatedSJ = TRUE,
  novelSplicing_useTJ = TRUE,
  overwrite = FALSE,
  n_threads = 1,
  lowMemoryMode = TRUE
)
```

Arguments

Experiment	(Required) A 2 or 3 column data frame, ideally generated by findSpliceWizOutput or findSamples . The first column designate the sample names, and the 2nd column contains the path to the processBAM output file (of type <code>sample.txt.gz</code>). (Optionally) a 3rd column contains the coverage files (of type <code>sample.cov</code>) of the corresponding samples. NB: all other columns are ignored.
reference_path	(Required) The path to the reference generated by Build-Reference-methods
output_path	(Required) The path to contain the output files for the collated dataset
IRMode	(default <code>SpliceOver</code>) The algorithm to calculate 'splice abundance' in IR quantification. Valid options are <code>SpliceOver</code> and <code>SpliceMax</code> . See details
packageCOVfiles	(default <code>FALSE</code>) Whether COV files should be copied over to the <code>NxtSE</code> object. This is useful if one wishes to transfer the <code>NxtSE</code> folder to a collaborator, who can then open the <code>NxtSE</code> object with valid COV file paths.

<code>novelSplicing</code>	(default FALSE) Whether <code>collateData</code> will use novel junction reads detected in samples to infer novel splice variants. All tandem split reads (those bridging two consecutive splice junctions) are used, as well as novel split reads that satisfy abundance criteria (see <code>novelSplicing_minSamples</code> , <code>novelSplicing_minSamplesAboveThreshold</code> , and <code>novelSplicing_countThreshold</code>) are used to synthesise a dataset-specific SpliceWiz reference. See details.
<code>forceStrandAgnostic</code>	(default FALSE) In poorly-prepared stranded libraries, it may be better to quantify in unstranded mode. Set this to TRUE if your stranded libraries may be contaminated with unstranded reads
<code>novelSplicing_minSamples</code>	(default 3) Novel junctions are included in building of novel reference if number samples with non-zero counts exceeds this number.
<code>novelSplicing_countThreshold</code>	(default 10) Threshold of split-reads across novel junctions; used in conjunction with <code>novelSplicing_minSamplesAboveThreshold</code>
<code>novelSplicing_minSamplesAboveThreshold</code>	(default 1) Novel junctions are included in building of novel reference if novel junction reads are above a pre-defined threshold exceeds this number
<code>novelSplicing_requireOneAnnotatedSJ</code>	(default TRUE) The default requires novel junctions to have one annotated splice site. If this is disabled, <code>collateData</code> will include novel junctions where neither splice site is annotated.
<code>novelSplicing_useTJ</code>	(default TRUE) For novel splicing, should SpliceWiz use reads with 2 or more junctions to find novel exons? Ignored if <code>novelSplicing</code> is set to FALSE.
<code>overwrite</code>	(default FALSE) If <code>collateData()</code> has previously been run using the same set of samples, it will not be overwritten unless this is set to TRUE.
<code>n_threads</code>	(default 1) The number of threads to use. If you run out of memory, try lowering the number of threads
<code>lowMemoryMode</code>	(default TRUE) <code>collateData()</code> will perform optimizations to conserve memory if this is set to TRUE. Otherwise, will prioritise performance.

Details

In Windows, `collateData` runs using only 1 thread, as BiocParallel's `MulticoreParam` is not supported.

It is assumed that all sample [processBAM](#) outputs were generated using the same reference.

The combination of junction counts and IR quantification from [processBAM](#) is used to calculate percentage spliced in (PSI) of alternative splice events, and intron retention ratios (IR-ratio) of retained introns. Also, QC information is collated. Data is organised in a H5file and FST files for memory and processor efficient downstream access using [makeSE](#).

The original IRFinder algorithm, see the following [wiki](#), uses SpliceMax to estimate abundance of spliced transcripts. This calculates the number of mapped splice events that share the boundary coordinate of either the left or right flanking exon `SpliceLeft`, `SpliceRight`, estimating splice abundance as the larger of the two values.

SpliceWiz proposes a new algorithm, SpliceOver, to account for the possibility that the major isoform shares neither boundary, but arises from either of the flanking exon clusters. Exon clusters are contiguous regions covered by exons from any transcript (except those designated as `retained_intron` or `sense_intronic`), and are separated by obligate intronic regions (genomic regions that are introns for all transcripts). For introns that are internal to a single exon cluster (i.e. akin to "known-exon" introns from IRFinder), SpliceOver uses `GenomicRanges::findOverlaps` to sum all splice reads that overlap the same genomic region as the intron of interest.

Detection of novel ASEs: When `novelSplicing` is set to `TRUE`, novel junctions (split reads across unannotated junctions from samples of the dataset being collated) are used in conjunction with the reference to compile a list of novel ASEs. To avoid being overwhelmed by a large number of false positive novel junctions (often due to mis-alignments), a simple filtering strategy is used. This involves including novel junctions only if it occurs in a minimum number of samples (default 3), or if the number of split reads of a novel junction is above a pre-defined threshold (default 10) in a certain number of samples (default 1). These parameters can be set using `novelSplicing_minSamples`, `novelSplicing_countThreshold` and `novelSplicing_minSamplesAboveThreshold` respectively.

Value

`collateData()` writes to the directory given by `output_path`. This output directory is portable (i.e. it can be moved to a different location after running `collateData()` before running `makeSE`), but individual files within the output folder should not be moved.

Also, the `processBAM` and `collateData` output folders should be copied to the same destination and their relative paths preserved. Otherwise, the locations of the "COV" files will not be recorded in the collated data and will have to be re-assigned using `covfile(se)<-`. See `makeSE`

See Also

[processBAM](#), [makeSE](#)

Examples

```
buildRef(
  reference_path = file.path(tempdir(), "Reference"),
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)

bams <- SpliceWiz_example_bams()
processBAM(bams$path, bams$sample,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "SpliceWiz_Output")
)

expr <- findSpliceWizOutput(file.path(tempdir(), "SpliceWiz_Output"))
collateData(expr,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "Collated_output")
)
```

```
# Enable novel splicing:

collateData(expr,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "Collated_output"),
  novelSplicing = TRUE
)
```

`coord2GR`*Converts genomic coordinates into a GRanges object*

Description

This function takes a string vector of genomic coordinates and converts it into a GRanges object.

Usage

```
coord2GR(coordinates)
```

Arguments

`coordinates` A string vector of one or more genomic coordinates to be converted

Details

Genomic coordinates can take one of the following syntax:

- `seqnames:start`
- `seqnames:start-end`
- `seqnames:start-end/strand`

The following examples are considered valid genomic coordinates:

- `"chr1:21535"`
- `"chr3:10550-10730"`
- `"X:51231-51330/-"`
- `"chrM:2134-5232/+"`

Value

A GRanges object that corresponds to the given coordinates

Examples

```
se <- SpliceWiz_example_NxtSE()

coordinates <- rowData(se)$EventRegion

gr <- coord2GR(coordinates)
```

covDataObject-class *Container to hold raw data for SpliceWiz coverage plots*

Description

This object is generated using `getCoverageData` or `getGenomeData` methods, and is used as input for generating coverage plots.

Usage

```
## S4 method for signature 'covDataObject'  
showEvents(object)
```

```
getCoverageData(  
  se,  
  Event,  
  Gene,  
  seqname,  
  start,  
  end,  
  coordinates,  
  strand = c("*", "+", "-"),  
  zoom_factor = 0.2,  
  bases_flanking = 100,  
  tracks,  
  condition,  
  ...  
)
```

```
getGenomeData(  
  reference_path,  
  Gene,  
  seqname,  
  start,  
  end,  
  coordinates,  
  zoom_factor = 0.2,  
  bases_flanking = 100,  
  ...  
)
```

```
plotAnnoTrack(  
  object,  
  Event,  
  view_start,  
  view_end,  
  reverseGenomeCoords = FALSE,
```



```

    condensed = FALSE,
    selected_transcripts = "",
    plot_key_isoforms = FALSE,
    usePlotly = FALSE,
    ...
)

```

Arguments

object	For plotAnnoTrack(), the covDataObject created by getCoverageData() or getGenomeData()
se	A NxtSE object, created by makeSE . COV files must be linked to the NxtSE object. To do this, see the example in makeSE . Required by plotCoverage. Not required by plotGenome if reference_path is supplied.
Event	The EventName of the IR / alternative splicing event to be displayed. Use rownames(se) to display a list of valid events.
Gene	Whether to use the range for the given Gene. If given, overrides Event (but Event or norm_event will be used to normalise by condition). Valid Gene entries include gene_id (Ensembl ID) or gene_name (Gene Symbol).
seqname, start, end	The chromosome (string) and genomic start/end coordinates (numeric) of the region to display. If present, overrides both Event and Gene. E.g. for a given region of chr1:10000-11000, use the parameters: seqname = "chr1", start = 10000, end = 11000
coordinates	A string specifying genomic coordinates can be given instead of seqname, start, end. Must be of the format "chr:start-end", e.g. "chr1:10000-11000"
strand	Whether to show coverage of both strands "*" (default), or from the "+" or "-" strand only.
zoom_factor	Zoom out from event. Each level of zoom zooms out by a factor of 3. E.g. for a query region of chr1:10000-11000, if a zoom_factor of 1.0 is given, chr1:99000-12000 will be displayed.
bases_flanking	(Default = 100) How many bases flanking the zoomed window. Useful when used in conjunction with zoom_factor == 0. E.g. for a given region of chr1:10000-11000, if zoom_factor = 0 and bases_flanking = 100, the region chr1:9900-11100 will be displayed.
tracks	The names of individual samples, or the names of the different conditions to be plotted. For the latter, set condition to the specified condition category.
condition	To display normalised coverage per condition, set this to the condition category. If omitted, tracks are assumed to refer to the names of individual samples.
...	Ignored / not used
reference_path	The path of the reference generated by Build-Reference-methods . Required by plotGenome if a NxtSE object is not specified.
view_start, view_end	Start and end coordinates of plotting function. Note that plot coordinates may be different from retrieval coordinates and is useful for zooming in.

reverseGenomeCoords	Whether the genomic axis should be reversed to make it more convenient to plot reverse-stranded genes
condensed	(default 'FALSE') Whether the genomic track should be condensed to plot whole genes, rather than transcripts. Preferred if multiple genes are plotted on a zoomed-out plot
selected_transcripts	(default "") One or more transcript names or ID's to be displayed on the annotation track.
plot_key_isoforms	(default FALSE) If TRUE, plots only transcripts involved in the given splicing Event.
usePlotly	(default FALSE) Whether to return a plotly or ggplot object.

Value

For `getCoverageData()`: A `covDataObject` containing required data used to generate downstream
 For `plotAnnoTrack()`: A `ggplot` or `plotly` object

Functions

- `showEvents(covDataObject)`: Returns the `EventNames` for which events can be normalized using the given `covDataObject`
- `getCoverageData()`: Get coverage / genome data for plotting coverage plots
- `getGenomeData()`: Get coverage / genome data for plotting coverage plots
- `plotAnnoTrack()`: Directly plots the annotation from a `covDataObject`.

See Also

[covPlotObject](#)

Examples

```
se <- SpliceWiz_example_NxtSE(novelSplicing = TRUE)

# Assign annotation of the experimental conditions
colData(se)$treatment <- rep(c("A", "B"), each = 3)

dataObj <- getCoverageData(
  se,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
  tracks = colnames(se)
)

# Show `EventName`s of supported splicing events
# contained within covDataObject

showEvents(dataObj)
```

```

# A limited covDataObject containing only the reference can be generated
# from the SpliceWiz reference

buildRef(
  reference_path = file.path(tempdir(), "Reference"),
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)

genomeObj <- getGenomeData(
  reference_path = file.path(tempdir(), "Reference"),
  Gene = "SRSF3"
)

# Plot reference track directly from the covDataObject

# NB: Event plotting is not supported for reference-derived `covDataObject`s
plotAnnoTrack(genomeObj)

plotAnnoTrack(dataObj, Event = "SE:SRSF3-203-exon4;SRSF3-202-int3")

```

Coverage

Calls SpliceWiz's C++ function to retrieve coverage from a COV file

Description

This function returns an RLE / RLEList or data.frame containing coverage data from the given COV file

COV files are generated by SpliceWiz's [processBAM](#) and [BAM2COV](#) functions. It records alignment coverage for each nucleotide in the given BAM file. It stores this data in "COV" format, which is an indexed BGZF-compressed format specialised for the storage of unstranded and stranded alignment coverage in RNA sequencing.

Unlike BigWig files, COV files store coverage for both positive and negative strands.

These functions retrieves coverage data from the specified COV file. They are computationally efficient as they utilise random-access to rapidly search for the requested data from the COV file.

Usage

```

getCoverage(file, seqname = "", start = 0, end = 0, strand = c("*", "+", "-"))

getCoverage_DF(
  file,
  seqname = "",

```

```

    start = 0,
    end = 0,
    strand = c("*", "+", "-")
  )

  getCoverageRegions(
    file,
    regions,
    strandMode = c("unstranded", "forward", "reverse")
  )

  getCoverageBins(
    file,
    region,
    bins = 2000,
    strandMode = c("unstranded", "forward", "reverse"),
    bin_size
  )

```

Arguments

file	(Required) The file name of the COV file
seqname	(Required for <code>getCoverage_DF</code>) A string denoting the chromosome name. If left blank in <code>getCoverage</code> , retrieves RLEList containing coverage of the entire file.
start,end	1-based genomic coordinates. If <code>start = 0</code> and <code>end = 0</code> , will retrieve RLE of specified chromosome.
strand	Either "*", "+", or "-"
regions	A GRanges object for a set of regions to obtain mean / total coverage from the given COV file.
strandMode	The stranded-ness of the RNA-seq experiment. "unstranded" means that an unstranded protocol was used. Stranded protocols can be either "forward", where the first read is the same strand as the expressed transcript, or "reverse" where the second strand is the same strand as the expressed transcript.
region	In <code>getCoverageBins</code> , a single query region as a GRanges object
bins	In <code>getCoverageBins</code> , the number of bins to divide the given region. If <code>bin_size</code> is given, overrides this parameter
bin_size	In <code>getCoverageBins</code> , the number of nucleotides per bin

Value

For `getCoverage`: If `seqname` is left as "", returns an RLEList of the whole BAM file, with each RLE in the list containing coverage data for one chromosome. Otherwise, returns an RLE containing coverage data for the requested genomic region

For `getCoverage_DF`: Returns a two-column data frame, with the first column coordinate denoting genomic coordinate, and the second column value containing the coverage depth for each coordinate nucleotide.

For `getCoverageRegions`: Returns a `GRanges` object with an extra metacolumn: `cov_mean`, which gives the mean coverage of each of the given ranges.

For `getCoverageBins`: Returns a `GRanges` object which spans the given region, divided by the number of bins or by width as given by `bin_size`. Mean coverage in each bin is calculated (returned by the `cov_mean` metadata column). This function is useful for retrieving coverage of a large region for visualisation, especially when the size of the region vastly exceeds the width of the figure.

Functions

- `getCoverage()`: Retrieves alignment coverage as an `RLE` or `RLElist`
- `getCoverage_DF()`: Retrieves alignment coverage as a `data.frame`
- `getCoverageRegions()`: Retrieves total and mean coverage of a `GRanges` object from a COV file
- `getCoverageBins()`: Retrieves coverage of a single region from a COV file, binned by the given number of bins or `bin_size`

Examples

```
se <- SpliceWiz_example_NxtSE()

cov_file <- covfile(se)[1]

# Retrieve Coverage as RLE

cov <- getCoverage(cov_file, seqname = "chrZ",
  start = 10000, end = 20000,
  strand = "*"
)

# Retrieve Coverage as data.frame

cov.df <- getCoverage_DF(cov_file, seqname = "chrZ",
  start = 10000, end = 20000,
  strand = "*"
)

# Retrieve mean coverage of 100-nt window regions as defined
# in a GRanges object:

gr <- GenomicRanges::GRanges(
  seqnames = "chrZ",
  ranges = IRanges::IRanges(
    start = seq(1, 99901, by = 100),
    end = seq(100, 100000, by = 100)
  ), strand = "-"
)

gr.unstranded <- getCoverageRegions(cov_file,
  regions = gr,
  strandMode = "unstranded"
```

```

)

gr.stranded <- getCoverageRegions(cov_file,
  regions = gr,
  strandMode = "reverse"
)

# Retrieve binned coverage of a large region

gr.fetch <- getCoverageBins(
  cov_file,
  region = GenomicRanges::GRanges(seqnames = "chrZ",
    ranges = IRanges::IRanges(start = 100, end = 100000),
    strand = "*"
  ),
  bins = 2000
)

# Plot coverage using ggplot:

require(ggplot2)

ggplot(cov.df, aes(x = coordinate, y = value)) +
  geom_line() + theme_white

ggplot(as.data.frame(gr.unstranded),
  aes(x = (start + end) / 2, y = cov_mean)) +
  geom_line() + theme_white

ggplot(as.data.frame(gr.fetch),
  aes(x = (start + end)/2, y = cov_mean)) +
  geom_line() + theme_white

# Export COV data as BigWig

cov_whole <- getCoverage(cov_file)
bw_file <- file.path(tempdir(), "sample.bw")
rtracklayer::export(cov_whole, bw_file, "bw")

```

covPlotly-class

Container for plotly-based coverage plots

Description

A covPlotly object is created when `plotView` is called using a covPlotObject as input. It stores metadata alongside the plotly object, which allows it to be drawn at various resolutions. Smaller resolutions lead to faster draws at expense of more jagged plots.

Usage

```
## S4 method for signature 'covPlotly'
getExonRanges(object)

## S4 method for signature 'covPlotly'
setResolution(object, resolution)

## S4 method for signature 'covPlotly'
showExons(object)
```

Arguments

object	A covPlotly object
resolution	How many horizontal pixels of resolution should be shown in the final plotly object. Set to 0 to disable.

Value

For show(): A plotly object synthesised by plotView() For getExonRanges(): A named GRanges object containing exon ranges For showExons(): A named GRanges object containing exon ranges, and additionally "shows" the plotly coverage plot with annotation replaced by named exons For setResolution() Returns the covPlotly object with addition of resolution set by the corresponding parameter. When show() is called, the plotly object with the new coverage resolution will be displayed.

Functions

- getExonRanges(covPlotly): Returns a named GRanges object containing exon ranges, without showing the associated plotly object
- setResolution(covPlotly): Returns a covPlotly object after setting the output resolution of the plotly-based coverage plots.
- showExons(covPlotly): Returns a named GRanges object containing exon ranges, and shows the plotly object with the annotation track showing the named exons

See Also

[plotView](#)

Examples

```
se <- SpliceWiz_example_NxtSE(novelSplicing = TRUE)

# Assign annotation of the experimental conditions
colData(se)$treatment <- rep(c("A", "B"), each = 3)

# Retrieve coverage data for all samples for the gene "SRSF3" (and surrounds)

dataObj <- getCoverageData(
  se,
```

```

    Gene = "SRSF3",
    tracks = colnames(se)
  )

plotObj_samples <- getPlotObject(
  dataObj,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3"
)

if(interactive()) {
  # Create covPlotly object by setting `usePlotly = TRUE`
  p <- plotView(plotObj_samples, usePlotly = TRUE)

  # Display plotly plot
  show(p)

  # Set resolution to 2000; display new plot
  p <- setResolution(p, resolution = 2000)
  show(p)

  # Display exon annotation along with generated plot;
  # - also returns GRanges object
  gr <- showExons(p)
}

```

covPlotObject-class *Versatile coverage plots for SpliceWiz*

Description

Here, we implement fast and versatile ggplot and plotly based coverage and sashimi plots. Users can plot with unlimited number of individual, individual-normalized, or group-normalized tracks. Also implemented is user-defined group-comparison differential plots (including t-test plots). Additionally, users can generate ggplots subsetted by exon groups. See details below.

Usage

```

getPlotObject(object, Event, strand = c("*", "+", "-"), tracks, condition, ...)

## S4 method for signature 'covPlotObject'
tracks(object)

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

plotView(
  object,

```



```

view_start,
view_end,
oldP = covPlotly(),
centerByEvent = FALSE,
EventZoomFactor = 0.2,
EventBasesFlanking = 100,
resolution = 5000,
trackList = list(),
diff_stat = c("t-test", "none"),
diffList = list(),
reverseGenomeCoords = FALSE,
ribbon_mode = c("sd", "sem", "ci", "none"),
normalizeCoverage = FALSE,
plotAnnotations = TRUE,
plotAnnoSubTrack = TRUE,
showExonRanges = FALSE,
verticalLayout = c(4, 1, 1, 2),
horizontalLayout = c(),
filterByTranscripts = "",
filterByEventTranscripts = FALSE,
filterByExpressedTranscripts = TRUE,
condenseTranscripts = FALSE,
plotJunctions = TRUE,
plotJuncPSI = FALSE,
junctionThreshold = 0.01,
plotRanges = GRanges(),
rangesBasesFlanking = 100,
usePlotly = FALSE,
...
)

```

Arguments

object	For <code>getPlotObject()</code> , a <code>covDataObject</code> created using <code>getCoverageData</code> . For <code>plotView()</code> , a <code>covPlotObject</code> created using <code>getPlotObject()</code> .
Event	The <code>EventName</code> of the alternative splicing event which will be highlighted and used for normalization
strand	The strand for coverage / junction plotting. Options are "+", "-", or "*" (unstranded - default)
tracks	Sample names or condition categories
condition	For condition-based group plots, the name of the condition.
...	Ignored / not used
view_start, view_end	The start and end coordinates for plotting
oldP	(Optional) If plotting the same tracks and track-widths, supplying the old <code>covPlotly</code> object (returned from a previous call to <code>plotView()</code>) results in faster run-time (as <code>plotly::subplot</code> is a time-consuming function)

centerByEvent	(default FALSE) If true, centers the view to the specified Event
EventZoomFactor	If centerByEvent = TRUE, the zoom-out factor to plot the view. Zooms out in exponents of 3 (i.e., zoom of 1 means 3x, 2 means 9x, and 0 means 1x)
EventBasesFlanking	(default 100) If centerByEvent = TRUE, includes how many bases flanking the event.
resolution	The number of horizontal "pixels" or data-points to plot. This is calculated per sub-plot. Smaller numbers lead to lower resolution but faster plots. Default is 5000
trackList	A list, with each element being a vector of 1 or more track names or indices to plot. If a vector is supplied it will be coerced to a list
diff_stat	(default "t-test") Which statistical method to perform differential comparisons.
diffList	A list, with each element being a vector of size 2, containing names or indices of which tracks to contrast.
reverseGenomeCoords	If TRUE, the genomic coordinate axis will be reversed to plot negative stranded genes
ribbon_mode	The statistic to represent variance. Options are "sd" - standard deviation, "sem" - standard error of the mean, "ci" - 95% confidence interval, or "none"
normalizeCoverage	If TRUE, coverages and junctions of individual samples will be normalized by the given Event.
plotAnnotations	Whether the main annotation track should be plotted
plotAnnoSubTrack	If plotting by exon ranges (using plotRanges), whether a separate sub-track showing zoomed-in exons should be shown above the main annotation track (and below the coverage plots)
showExonRanges	(only applies if usePlotly = FALSE) Whether the main annotation track should be replaced by labeled exon names. If TRUE the returned value of plotView() is a named GRanges object containing the exon ranges
verticalLayout	A vector (of length 4) containing relative heights of the following elements: (1) main block of coverage tracks, (2) differential track, (3) annotation sub-track, and (4) main annotation track. Default c(4, 1, 1, 2)
horizontalLayout	A vector containing relative widths of coverage tracks. Only used alongside plotRanges with more than 1 range to plot. If omitted, plotView will attempt to scale widths to the widths of the exon ranges.
filterByTranscripts	(default "") One or more named transcripts to filter the annotation track.
filterByEventTranscripts	(default FALSE) If TRUE, only transcripts involved in the given Event will be plotted, if any

filterByExpressedTranscripts	(default TRUE) Only transcripts with supported junctions will be plotted on the annotation axis. An expressed junction is that which contains more than the minimum junctionThreshold in at least 1 track
condenseTranscripts	Whether to plot by genes TRUE or transcripts FALSE
plotJunctions	Whether to plot junction counts as numbered arcs. Plots normalized junctions if normalizeCoverage = TRUE.
plotJuncPSI	If plotting group coverage plots, whether to plot mean +/- sd of normalized junction counts FALSE, or estimated junction PSI based on SpliceOver metric applied to each junction TRUE.
junctionThreshold	(default 0.01) Junctions with expressions below this threshold will not be plotted. For raw counts, this is a fraction of maximum coverage value of the track.
plotRanges	A GRanges object containing one or more exon ranges to plot. If given, view_start and view_end will be ignored. Typical use is to use the output of the plotView(..., usePlotly = FALSE), which returns a named GRanges object, then subset this output by exon name.
rangesBasesFlanking	(default 100) How many flanking bases to add to each of plotRanges. Ignored if only 1 range given (or using view_start and view_end)
usePlotly	If TRUE, returns a covPlotly object containing the plotly-based interactive plot. If FALSE, returns a ggplot object.

Details

The typical pipeline for plotting versatile coverage plots is as follows:

- A covDataObject is generated by calling `getCoverageData()` using an input `NxtSE` object. This step retrieves coverage, junction counts and normalization data for the relevant genomic region being queried. A new `covDataObject` is necessary when querying a new genomic region.
- A `covPlotObject` is generated by calling `getPlotObject()` using an input `covDataObject`. This step retrieves alternative splicing event specific data, such as normalized coverages, or group combined coverages. A new `covPlotObject` is required when changing condition, Event, strand, or when querying using a different set of tracks.
- Plots can be generated by calling `plotView()` using a `covPlotObject`. Interactive plotly plots can be generated by setting `usePlotly = TRUE`, otherwise, static plots are generated. For interactive plots, a `covPlotly` object is returned, which contains raw data which is downsampled by pixel resolution prior to plotting for performance reasons. A new `covPlotly` is required unless one only wishes to downsample the resolution
 - see [setResolution](#) for `covPlotly` objects.

Tracks are now versatile (unlimited). Samples are retrieved by individual sample names at `getCoverageData()`. If condition is set in `getPlotObject()`, track names are defined by their condition categorical names; otherwise, tracks are named by individual samples when retrieved using `getPlotObject()`.

- When calling `plotView()`, `trackList` by default displays all tracks as ordered in the `covPlotObject`. Users can supply a vector containing either the track names (or numbers, as ordered in the `covPlotObject`). Alternatively, multiple traces can be stacked in a single track by using a list, e.g. `trackList = list(A = c(1,2,3), B = c(4,5,6))`.
- For differential comparisons, `diffList` takes a list of pairs of samples. For example, if `trackList = list("A", "B")`, then setting `diffList = list(c("A", "B"))` will compare groups "A" and "B". This is only activated by setting `diff_stat` to anything other than none. For now, only t-test is supported.

`plotView()` supports plotting by exon ranges, for which only static plots are currently supported. The workflow for generating such a plot is as follows:

- A `GRanges` object is returned by the `plotView()` function and setting `showExonRanges = TRUE`. `plotView()` will simultaneously show an annotation plot of exons labelled by their "exon names", which is the transcript name appended with "-E" followed by the exon number.
- If `plotView()` is called and `usePlotly = TRUE` is set, a `covPlotly` object is returned. Calling `showExons()` on this object will display a plotly plot showing exon names, and returning a `GRanges` object of exon ranges.
- Exon ranges can be supplied to the `plotView()` function by setting the `plotRanges` parameter as a `GRanges` object. This will generate a static plot showing coverage plots segmented by exons.

Value

For `getPlotObject()`: A `covPlotObject` object containing Event-based data to create coverage plots using `plotView()`.

For `plotView()`:

- If `usePlotly = TRUE`, returns a `covPlotly` object containing plotly-based interactive plot
- If `usePlotly = FALSE`, returns a patchwork-assembled static plot, unless `showExonRanges = TRUE` in which it shows the plot and returns a named `GRanges` object containing exon ranges.

Functions

- `getPlotObject()`: Generates a `covPlotObject` object from a `covDataObject`. Allows users to change parameters such as viewing window, conditions, tracks, and other parameters, for customizing plot parameters
- `tracks(covPlotObject)`: Returns the tracks contained in the `covPlotObject` object
- `condition(covPlotObject)`: Returns the condition value set in the `covPlotObject` object
- `plotView()`: Creates a coverage plot using the stored data in the `covPlotObject`

See Also

[getCoverageData covPlotly](#)

Examples

```
se <- SpliceWiz_example_NxtSE(novelSplicing = TRUE)

# Assign annotation of the experimental conditions
colData(se)$treatment <- rep(c("A", "B"), each = 3)

# Retrieve coverage data for all samples for the gene "SRSF3" (and surrounds)

dataObj <- getCoverageData(
  se,
  Gene = "SRSF3",
  tracks = colnames(se)
)

# Retrieves raw / normalized coverage / junction data for the
# specified SRSF3 skipped exon event:

plotObj_samples <- getPlotObject(
  dataObj,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3"
)

# Retrieves data for samples grouped by the specified condition

plotObj_group <- getPlotObject(
  dataObj,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
  condition = "treatment",
  tracks = c("A", "B")
)

# Display tracks and conditions of covPlotObject

tracks(plotObj_group)
condition(plotObj_group)

# Show static ggplots

plotView(plotObj_samples)

plotView(plotObj_group, centerByEvent = TRUE)

# Plot junctions using PSI estimates based on individual junction SpliceOver
# metrics

plotView(plotObj_group, centerByEvent = TRUE, plotJuncPSI = TRUE)

# Show normalized coverages, individual samples stacked in grouped tracks

plotView(
  plotObj_samples,
```

```

    normalizeCoverage = TRUE,
    trackList = list(A = c(1,2,3), B = c(4,5,6))
  )

# Show stacked group comparisons with t-test

plotView(
  plotObj_group,
  trackList = list(c(1,2)),
  diffList = list(c("A", "B")),
  diff_stat = "t-test"
)

# Show interactive plotly:

if(interactive()) {
  p <- plotView(plotObj_samples, usePlotly = TRUE)
  show(p)
}

# Show exons with coverage plot

# static:
gr <- plotView(plotObj_samples, showExonRanges = TRUE)

# interactive:
if(interactive()) {
  p <- plotView(plotObj_samples, usePlotly = TRUE)
  gr <- showExons(p)
}

# Plot coverage by exons

p <- plotView(plotObj_samples,
  plotRanges = gr[c("SRSF3-203-E3", "SRSF3-203-E4", "SRSF3-203-E5")],
  horizontalLayout = c(1,1,1)
)

```

example-SpliceWiz-data

SpliceWiz Example BAMs and NxtSE Experiment Object

Description

`SpliceWiz_example_bams()` is a wrapper function to obtain and make a local copy of 6 example files provided by the `NxtIRFdata` companion package to demonstrate the use of `SpliceWiz`. See [NxtIRFdata::example_bams](#) for a description of the provided BAM files.

`SpliceWiz_example_NxtSE()` retrieves a ready-made functioning `NxtSE` object. The steps to reproduce this object is shown in the example code in [makeSE](#)

Usage

```
SpliceWiz_example_bams()

SpliceWiz_example_NxtSE(novelSplicing = FALSE)
```

Arguments

`novelSplicing` Whether to import an example NxtSE with novel splice event discovery.

Value

In `SpliceWiz_example_bams()`: returns a 2-column data frame containing sample names and BAM paths of the example dataset.

In `SpliceWiz_example_NxtSE()`: returns a [NxtSE](#) object.

Functions

- `SpliceWiz_example_bams()`: Returns a 2-column data frame, containing sample names and sample paths (in `tempdir()`) of example BAM files
- `SpliceWiz_example_NxtSE()`: Returns a (in-memory / realized) NxtSE object that was pre-generated using the SpliceWiz example reference and example BAM files

References

Generation of the mappability files was performed using SpliceWiz using a method analogous to that described in:

Middleton R, Gao D, Thomas A, Singh B, Au A, Wong JJ, Bomane A, Cosson B, Eyraas E, Rasko JE, Ritchie W. IRFinder: assessing the impact of intron retention on mammalian gene expression. *Genome Biol.* 2017 Mar 15;18(1):51. doi:[10.1186/s1305901711844](https://doi.org/10.1186/s1305901711844)

See Also

[makeSE](#)

Examples

```
# returns a data frame with the first column as sample names, and the
# second column as BAM paths

SpliceWiz_example_bams()

# Returns a NxtSE object created by the example bams aligned to the
# mock NxtSE reference

se <- SpliceWiz_example_NxtSE()
```

 findSamples

Convenience Function to (recursively) find all files in a folder.

Description

Often, files e.g. raw sequencing FASTQ files, alignment BAM files, or [processBAM](#) output files, are stored in a single folder under some directory structure. They can be grouped by being in common directory or having common names. Often, their sample names can be gleaned by these common names or the names of the folders in which they are contained. This function (recursively) finds all files and extracts sample names assuming either the files are named by sample names (`level = 0`), or that their names can be derived from the parent folder (`level = 1`). Higher `level` also work (e.g. `level = 2`) mean the parent folder of the parent folder of the file is named by sample names. See details section below.

Usage

```
findSamples(sample_path, suffix = ".txt.gz", level = 0)

findFASTQ(
  sample_path,
  paired = TRUE,
  fastq_suffix = c(".fastq", ".fq", ".fastq.gz", ".fq.gz"),
  level = 0
)

findBAMS(sample_path, level = 0)

findSpliceWizOutput(sample_path, level = 0)
```

Arguments

<code>sample_path</code>	The path in which to recursively search for files that match the given suffix
<code>suffix</code>	A vector of or more strings that specifies the file suffix (e.g. <code>'bam'</code> denotes BAM files, whereas <code>".txt.gz"</code> denotes gzipped txt files).
<code>level</code>	Whether sample names can be found in the file names themselves (<code>level = 0</code>), or their parent directory (<code>level = 1</code>). Potentially parent of parent directory (<code>level = 2</code>). Support max <code>level <= 3</code> (for sanity).
<code>paired</code>	Whether to expect single FASTQ files (of the format <code>"sample.fastq"</code>), or paired files (of the format <code>"sample_1.fastq"</code> , <code>"sample_2.fastq"</code>)
<code>fastq_suffix</code>	The name of the FASTQ suffix. Options are: <code>".fastq"</code> , <code>".fastq.gz"</code> , <code>".fq"</code> , or <code>".fq.gz"</code>

Details

Paired FASTQ files are assumed to be named using the suffix `_1` and `_2` after their common names; e.g. `sample_1.fastq`, `sample_2.fastq`. Alternate FASTQ suffixes for `findFASTQ()` include `".fq"`, `".fastq.gz"`, and `".fq.gz"`.

In BAM files, often the parent directory denotes their sample names. In this case, use `level = 1` to automatically annotate the sample names using `findBAMS()`.

`processBAM` outputs two files per BAM processed. These are named by the given sample names. The text output is named "sample1.txt.gz", and the COV file is named "sample1.cov", where `sample1` is the name of the sample. These files can be organised / tabulated using the function `findSpliceWizOutput`. The generic function `findSamples` will organise the `processBAM` text output files but exclude the COV files. Use the latter as the Experiment in `collateData` if one decides to collate an experiment without linked COV files, for portability reasons.

Value

A multi-column data frame with the first column containing the sample name, and subsequent columns being the file paths with suffix as determined by `suffix`.

Functions

- `findSamples()`: Finds all files with the given suffix pattern. Annotates sample names based on file or parent folder names.
- `findFASTQ()`: Use `findSamples()` to return all FASTQ files in a given folder
- `findBAMS()`: Use `findSamples()` to return all BAM files in a given folder
- `findSpliceWizOutput()`: Use `findSamples()` to return all `processBAM` output files in a given folder, including COV files

Examples

```
# Retrieve all BAM files in a given folder, named by sample names
bam_path <- tempdir()
example_bams(path = bam_path)
df.bams <- findSamples(sample_path = bam_path,
  suffix = ".bam", level = 0)
# equivalent to:
df.bams <- findBAMS(bam_path, level = 0)

# Retrieve all processBAM() output files in a given folder,
# named by sample names

expr <- findSpliceWizOutput(file.path(tempdir(), "SpliceWiz_Output"))
## Not run:

# Find FASTQ files in a directory, named by sample names
# where files are in the form:
# - "./sample_folder/sample1.fastq"
# - "./sample_folder/sample2.fastq"

findFASTQ("./sample_folder", paired = FALSE, fastq_suffix = ".fastq")

# Find paired gzipped FASTQ files in a directory, named by parent directory
# where files are in the form:
# - "./sample_folder/sample1/raw_1.fq.gz"
# - "./sample_folder/sample1/raw_2.fq.gz"
```

```
# - "./sample_folder/sample2/raw_1.fq.gz"
# - "./sample_folder/sample2/raw_2.fq.gz"

findFASTQ("./sample_folder", paired = TRUE, fastq_suffix = ".fq.gz")

## End(Not run)
```

Gene-ontology-methods *Gene ontology (over-representation) analysis using enriched genes of top alternative splicing events*

Description

Genes containing differential alternative splicing events (ASEs) may be enriched in key functional pathways. This can be identified using a simple over-representation analysis. Biologists can identify key pathways of interest in order to focus on studying ASEs belonging to genes of functional interest.

Usage

```
goASE(
  enrichedEventNames,
  universeEventNames = NULL,
  se,
  ontologyType = c("BP", "MF", "CC"),
  pAdjustMethod = c("BH", "holm", "hochberg", "hommel", "bonferroni", "BY", "fdr",
    "none"),
  ontologyRef = NULL,
  ...
)

goGenes(
  enrichedGenes,
  universeGenes = NULL,
  ontologyRef,
  ontologyType = c("BP", "MF", "CC"),
  pAdjustMethod = c("BH", "holm", "hochberg", "hommel", "bonferroni", "BY", "fdr",
    "none"),
  ...
)

extract_gene_ids_for_GO(enrichedEventNames, universeEventNames = NULL, se)

subset_EventNames_by_GO(EventNames, go_id, se)

plotGO(
```

```

res_go = NULL,
plot_x = c("log10FDR", "foldEnrichment", "nGenes"),
plot_size = c("nGenes", "foldEnrichment", "log10FDR"),
plot_color = c("foldEnrichment", "nGenes", "log10FDR"),
filter_n_terms = 20,
filter_padj = 1,
filter_pvalue = 1,
trim_go_term = 50
)

```

Arguments

enrichedEventNames	A vector of EventNames. This is typically one or more EventNames of differential ASEs
universeEventNames	A vector of EventNames, typically the EventNames of all ASEs that were tested. If left as NULL, all genes are considered background genes.
se	The NxtSE object containing the GO reference and the EventNames
ontologyType	One of either "BP" - biological pathways, "MF" - molecular function, or "CC" - cellular component.
pAdjustMethod	The method for p-value adjustment for FDR. See ?p.adjust
ontologyRef	A valid gene ontology reference. This can be generated either using viewGO(reference_path) or ref(se)\$ontology. This field is required for goGenes() and optional for goASE(). See details.
...	Additional arguments to be passed to fgsea::fora()
enrichedGenes	A vector of gene_id representing the list of enriched genes. To generate a list of valid gene_id, see viewGenes
universeGenes	(default NULL) A vector of gene_id representing the list of background genes.
EventNames, go_id	In subset_EventNames_by_GO(), a vector of ASE EventNames to subset against the given go_id.
res_go	For plotGO, the gene ontology results data object returned by the goASE() function.
plot_x, plot_size, plot_color	What parameters should be plotted on the x-axis, bubble-size, or bubble-color? Valid options are c("log10FDR", "foldEnrichment", "nGenes"). Defaults are "log10FDR", "nGenes", "foldEnrichment" for x-axis, bubble size/color, respectively
filter_n_terms	(default 20) How many top terms to plot.
filter_padj, filter_pvalue	Whether given GO results should be filtered by adjusted p value (FDR) or nominal p value, respectively, prior to plot
trim_go_term	(default 50) For long GO terms, description will be trimmed by first n characters, where trim_go_term = n

Details

Users can perform GO analysis using either the GO annotation compiled via building the SpliceWiz reference using `buildRef()`, or via a custom-supplied gene ontology annotation. This is done by supplying their own GO annotations as an argument to `ontologyRef`. This should be coerceable to a `data.frame` containing the following columns:

- `gene_id` Gene ID's matching that used by the SpliceWiz reference
- `go_id` Gene ontology ID terms, of the form `GO:XXXXXX`

Value

For `goASE()` and `goGenes()`, a data table containing the following:

- `go_id`: Gene ontology ID
- `go_term`: Gene ontology term
- `pval`: Raw p values
- `padj`: Adjusted p values
- `overlap`: Number of enriched genes (of enriched ASEs)
- `size`: Number of background genes (of background ASEs)
- `overlapGenes`: A list of `gene_id`'s from genes of enriched ASEs
- `expected`: The number of overlap genes expected by random

For `extract_gene_ids_for_GO()`, a list containing the following:

- `genes`: A vector of enriched `gene_ids`
- `universe`: A vector of background `gene_ids`

For `subset_EventNames_by_GO()`, a vector of all ASE `EventNames` belonging to the given gene ontology `go_id`

Functions

- `goASE()`: Performs over-representation gene ontology analysis using a given list of enriched / background ASEs
- `goGenes()`: Performs GO analysis given the set of enriched and (optionally) the background (universe) genes.
- `extract_gene_ids_for_GO()`: Produces a list containing enriched and universe `gene_ids` of given enriched and background ASE `EventNames`
- `subset_EventNames_by_GO()`: Returns a list of ASEs enriched in a given gene ontology category
- `plotGO()`: Produces a lollipop plot based on the given gene ontology results object

See Also

[Build-Reference-methods](#) on how to generate gene ontology annotations

Examples

```
# Generate example reference with `Homo sapiens` gene ontology

ref_path <- file.path(tempdir(), "Reference_withGO")
buildRef(
  reference_path = ref_path,
  fasta = chrZ_genome(),
  gtf = chrZ_gtf(),
  ontologySpecies = "Homo sapiens"
)

# Perform GO analysis using first 1000 genes
ontology <- viewGO(ref_path)
allGenes <- sort(unique(ontology$gene_id))

exampleGeneID <- allGenes[1:1000]
exampleBkgdID <- allGenes

go_df <- goGenes(
  enrichedGenes = exampleGeneID,
  universeGenes = exampleBkgdID,
  ontologyRef = ontology
)

# Plots the top 12 GO terms

plotGO(go_df, filter_n_terms = 12)

# Below example code of how to use output of differential ASEs for GO analysis
# It will not work with the example dataset because the reference must be
# either human / mouse, or a valid `ontologySpecies` given to buildRef()
# We hope the example code is simple enough to understand for users to adapt
# to their own workflows.

## Not run:

se <- SpliceWiz_example_NxtSE(novelSplicing = TRUE)

colData(se)$treatment <- rep(c("A", "B"), each = 3)

require("limma")
res_limma <- ASE_limma(se, "treatment", "A", "B")

# Perform gene ontology analysis of the first 10 differential ASEs

go_df <- goASE(
  enrichedEventNames = res_limma$EventName[1:10],
  universeEventNames = res_limma$EventName,
  se = se
)

# Return a list of all ASEs belonging to the top enriched category
```

```

GOsubset_EventName <- subset_EventNames_by_GO(
  EventNames = res_limma$EventName,
  go_id = go_df$go_id[1],
  se = se
)

# Return a list of all ASEs belonging to the top enriched category.
# - typically used if one wishes to export `gene_id` for use in other gene
#   ontology tools

gene_id_list <- extract_gene_ids_for_GO(
  enrichedEventNames = res_limma$EventName[1:10],
  universeEventNames = res_limma$EventName,
  se = se
)

## End(Not run)

```

Graphics-User-Interface

Launches the SpliceWiz Graphics User Interface (GUI) using Shiny Dashboard

Description

This function launches the SpliceWiz interactive app using Shiny Dashboard. This is (by default) a dialog window within the RStudio application with the resolution specified by the `res` parameter. Alternatively, setting `mode = "browser"` will launch a resizable browser window (using the default internet browser). The demo mode can be launched by setting `demo = TRUE`. See the [SpliceWiz Quick-Start](#) for a guide to using the SpliceWiz GUI.

Usage

```

spliceWiz(
  mode = c("dialog", "browser"),
  res = c("1080p", "720p", "960p", "1440p"),
  demo = FALSE
)

```

Arguments

<code>mode</code>	(default "dialog") "dialog" displays SpliceWiz in a dialog box with specified width and height. "browser" opens SpliceWiz in a browser-like resizable window.
<code>res</code>	(default "1080p") Sets width and height of the app to pre-defined dimensions. Possible options are "720p", "960p", "1080p", "1440p", which specifies the height of the app. All are displayed in aspect ratio 16x9

demo (default FALSE) If set to TRUE, SpliceWiz will place demo reference and BAM files into the temporary directory.

Value

Runs an interactive shinydashboard SpliceWiz app with the specified mode.

Functions

- spliceWiz(): Launches the SpliceWiz GUI

Examples

```
if(interactive()) {
  # Launches interactive ShinyDashboard SpliceWiz app as fixed-size dialog box
  # 1080p = 1920 x 1080 pixels
  spliceWiz(mode = "dialog", res = "1080p")

  # Launches interactive ShinyDashboard SpliceWiz app as browser window
  spliceWiz(mode = "browser")
}
```

isCOV	<i>Validates the given file as a valid COV file</i>
-------	---

Description

This function takes the path of a possible COV file and checks whether its format complies with that of the COV format defined by this package.

Usage

```
isCOV(coverage_files)
```

Arguments

coverage_files A vector containing the file names of files to be checked

Details

COV files are BGZF-compressed files. The first 4 bytes of the file must always be 'COV\1', distinguishing it from BAM or other files in BGZF format. This function checks whether the given file complies with this.

Value

TRUE if all files are valid COV files. FALSE otherwise

See Also

[processBAM collateData](#)

Examples

```
se <- SpliceWiz_example_NxtSE()

cov_files <- covfile(se)

isCOV(cov_files) # returns true if these are true COV files
```

makeSE

Imports a collated dataset into the R session as an NxtSE object

Description

Creates a [NxtSE](#) object from the data (that was collated using [collateData](#)). This object is used for downstream differential analysis of IR and alternative splicing events using [ASE-methods](#), data generation for visualization of scatter plots and heatmaps via [make_plot_data](#) methods, and coverage visualisation using [plotCoverage](#)

Usage

```
makeSE(
  collate_path,
  colData,
  RemoveOverlapping = TRUE,
  realize = FALSE,
  verbose = TRUE
)
```

Arguments

<code>collate_path</code>	(Required) The output path of collateData pointing to the collated data
<code>colData</code>	(Optional) A data frame containing the sample annotation information. The first column must contain the sample names. Omit <code>colData</code> to generate a <code>NxtSE</code> object of the whole dataset without any assigned annotations. Alternatively, if the names of only a subset of samples are given, then <code>makeSE()</code> will construct the <code>NxtSE</code> object based only on the samples given. The <code>colData</code> can be set later using colData
<code>RemoveOverlapping</code>	(default = TRUE) Whether to filter out overlapping IR events belonging to minor isoforms. See details.
<code>realize</code>	(default = FALSE) Whether to load all assay data into memory. See details
<code>verbose</code>	(default = TRUE) Whether loading messages are displayed

Details

makeSE retrieves the data collated by `collateData`, and initialises a `NxtSE` object. It references the required on-disk assay data using `DelayedArrays`, thereby utilising 'on-disk' memory to conserve memory usage.

For extremely large datasets, loading the entire data into memory may consume too much memory. In such cases, make a subset of the `NxtSE` object (e.g. subset by samples) before loading the data into memory (RAM) using `realize_NxtSE`. Alternatively supply a data frame to the `colData` parameter of the `makeSE()` function. Only samples listed in the first column of the `colData` data frame will be imported into the `NxtSE` object.

It should be noted that downstream applications of `SpliceWiz`, including `ASE-methods`, `plotCoverage`, are much faster if the `NxtSE` is realized. It is recommended to realize the `NxtSE` object before extensive usage.

If COV files assigned via `collateData` have been moved relative to the `collate_path`, the created `NxtSE` object will not be linked to any COV files and `plotCoverage` cannot be used. To reassign these files, a vector of file paths corresponding to all the COV files of the data set can be assigned using `covfile(se) <- vector_of_cov_files`. See the example below for details.

If `RemoveOverlapping = TRUE`, `makeSE` will remove introns that overlap other introns with higher junction read counts in the dataset. This means that `SpliceWiz` will assess a set of non-overlapping introns which belong to likely major isoforms, ensuring that overlapping IR events are not 'double-counted'.

NB: Since version 1.3.4, `SpliceWiz` has improved the algorithm of generating the set of non-overlapping introns (prior versions appear to generate sets of introns that still overlap). To use the prior algorithm for compatibility with prior analysis, set `RemoveOverlapping = FALSE`.

Value

A `NxtSE` object containing the compiled data in `DelayedArrays` (or as matrices if `realize = TRUE`), pointing to the assay data contained in the given `collate_path`

Examples

```
# The following code can be used to reproduce the NxtSE object
# that can be fetched with SpliceWiz_example_NxtSE()

buildRef(
  reference_path = file.path(tempdir(), "Reference"),
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)

bams <- SpliceWiz_example_bams()
processBAM(bams$path, bams$sample,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "SpliceWiz_Output")
)

expr <- findSpliceWizOutput(file.path(tempdir(), "SpliceWiz_Output"))
collateData(expr,
```

```

reference_path = file.path(tempdir(), "Reference"),
output_path = file.path(tempdir(), "Collated_output")
)

se <- makeSE(collate_path = file.path(tempdir(), "Collated_output"))

# "Realize" NxtSE object to load all H5 assays into memory:

se <- realize_NxtSE(se)

# If COV files have been removed since the last call to collateData()
# reassign them to the NxtSE object, for example:

covfile_path <- system.file("extdata", package = "SpliceWiz")
covfile_df <- findSamples(covfile_path, ".cov")

covfile(se) <- covfile_df$path

```

make_plot_data	<i>Construct data of percent-spliced-in (PSI) matrices and group-average PSIs</i>
----------------	---

Description

makeMatrix() constructs a matrix of PSI values of the given alternative splicing events (ASEs).

makeMeanPSI() constructs a table of "average" PSI values, with samples grouped by a number of given conditions (e.g. "group A" and "group B") of a given condition category (e.g. condition "treatment"). See details below.

Usage

```

makeMatrix(
  se,
  event_list,
  sample_list = colnames(se),
  method = c("PSI", "logit", "Z-score"),
  depth_threshold = 10,
  logit_max = 5,
  na.percent.max = 0.1
)

```

```

makeMeanPSI(
  se,
  event_list = rownames(se),
  condition,
  conditionList,

```

```

    depth_threshold = 10,
    logit_max = 10
)

```

Arguments

se	(Required) A NxtSE object generated by makeSE
event_list	A character vector containing the names of ASE events (as given by the EventName column of differential ASE results table generated by one of the ASE-methods , or the rownames of the NxtSE object)
sample_list	(default = colnames(se)) In makeMatrix() , a list of sample names in the given experiment to be included in the returned matrix
method	In makeMatrix() , the values to be returned (default = "PSI"). It can alternately be "logit" which returns logit-transformed PSI values, or "Z-score" which returns Z-score-transformed PSI values
depth_threshold	(default = 10) Samples with the number of reads supporting either included or excluded isoforms below this values are excluded
logit_max	PSI values close to 0 or 1 are rounded up/down to $\text{plogis}(-\text{logit_max})$ and $\text{plogis}(\text{logit_max})$, respectively. See details.
na.percent.max	(default = 0.1) The maximum proportion of values in the given dataset that were transformed to NA because of low splicing depth. ASE events where there are a higher proportion (default 10%) NA values will be excluded from the final matrix. Most heatmap functions will spring an error if there are too many NA values in any given row. This option caps the number of NA values to avoid returning this error.
condition	The name of the column containing the condition values in <code>colData(se)</code>
conditionList	A list (or vector) of condition values of which to calculate mean PSIs

Details

Note that this function takes the geometric mean of PSI, by first converting all values to $\text{logit}(\text{PSI})$, taking the average $\text{logit}(\text{PSI})$ values of each condition, and then converting back to PSI using inverse logit .

Samples with low splicing coverage (either due to insufficient sequencing depth or low gene expression) are excluded from calculation of mean PSIs. The threshold can be set using `depth_threshold`. Excluding these samples is appropriate because the uncertainty of PSI is high when the total included / excluded count is low. Note that events where all samples in a condition is excluded will return a value of `NaN`.

Using logit-transformed PSI values is appropriate because PSI values are bound to the (0,1) interval, and are often thought to be beta-distributed. The link function often used with beta-distributed models is the logit function, which is defined as $\text{logit}(x) = \text{function}(x) \log(x / (1 - x))$, and is equivalent to [stats::qlogis](#). Its inverse is equivalent to [stats::plogis](#).

Users wishing to calculate arithmetic means of PSI are advised to use [makeMatrix](#), followed by [rowMeans](#) on subsetted sample columns.

Value

For `makeMatrix`: A matrix of PSI (or alternate) values, with columns as samples and rows as ASE events.

For `makeMeanPSI`: A 3 column data frame, with the first column containing `event_list` list of ASE events, and the last 2 columns containing the average PSI values of the nominator and denominator conditions.

Functions

- `makeMatrix()`: constructs a matrix of PSI values of the given alternative splicing events (ASEs)
- `makeMeanPSI()`: constructs a table of "average" PSI values

Examples

```
se <- SpliceWiz_example_NxtSE()

colData(se)$treatment <- rep(c("A", "B"), each = 3)

event_list <- rowData(se)$EventName

mat <- makeMatrix(se, event_list[1:10])

diag_values <- makeMeanPSI(se, event_list,
  condition = "treatment",
  conditionList = list("A", "B")
)
```

Mappability-methods *Calculate low mappability genomic regions*

Description

These functions empirically calculate low-mappability (Mappability Exclusion) regions using the given reference. A splice-aware alignment software capable of aligning reads to the genome is required. See details and examples below.

Usage

```
generateSyntheticReads(
  reference_path,
  read_len = 70,
  read_stride = 10,
  error_pos = 35,
  verbose = TRUE,
  alt_fasta_file
)
```

```

calculateMappability(
  reference_path,
  aligned_bam = file.path(reference_path, "Mappability", "Aligned.out.bam"),
  threshold = 4,
  n_threads = 1
)

```

Arguments

reference_path	The directory of the reference prepared by getResources
read_len	The nucleotide length of the synthetic reads
read_stride	The nucleotide distance between adjacent synthetic reads
error_pos	The position of the procedurally-generated nucleotide error from the start of each synthetic reads
verbose	Whether additional status messages are shown
alt_fasta_file	(Optional) The path to the user-supplied genome fasta file, if different to that found inside the resource subdirectory of the reference_path. If getResources has already been run, this parameter should be omitted.
aligned_bam	The BAM file of alignment of the synthetic reads generated by <code>generateSyntheticReads()</code> . Users should use a genome splice-aware aligner, preferably the same aligner used to align the samples in their experiment.
threshold	Genomic regions with this alignment read depth (or below) in the aligned synthetic read BAM are defined as low mappability regions.
n_threads	The number of threads used to calculate mappability exclusion regions from aligned bam file of synthetic reads.

Details

Creating a Mappability Exclusion BED file is a three-step process.

- First, using `generateSyntheticReads()`, synthetic reads are systematically generated using the given genome contained within `reference_path`, prepared via [getResources](#). Alternatively, use `alt_fasta_file` to set the genome sequence if this is different to that prepared by `getResources` or if `getResources` is not yet run.
- Second, an aligner such as STAR (preferably the same aligner used for the subsequent RNA-seq experiment) is required to align these reads to the source genome. Poorly mapped regions of the genome will be reflected by regions of low coverage depth.
- Finally, the BAM file containing the aligned reads is analysed using `calculateMappability()`, to identify low-mappability regions to compile the Mappability Exclusion BED file.

It is recommended to leave all parameters to their default settings. Regular users should only specify `reference_path`, `aligned_bam` and `n_threads`, as required.

NB: [STAR_mappability](#) runs all 3 steps required, using the STAR aligner. This only works in systems where STAR is installed.

NB2: [buildFullRef](#) builds the STAR reference, then calculates mappability. It then uses the calculated mappability regions to build the SpliceWiz reference.

NB3: In systems where STAR is not available, consider using HISAT2 or Rsubread. A working example using Rsubread is shown below.

Value

- For `generateSyntheticReads`: writes `Reads.fa` to the `Mappability` subdirectory inside the given `reference_path`.
- For `calculateMappability`: writes a gzipped BED file named `MappabilityExclusion.bed.gz` to the `Mappability` subdirectory inside `reference_path`. This BED file is automatically used by [buildRef](#) if its `MappabilityRef` parameter is not specified.

Functions

- `generateSyntheticReads()`: Generates synthetic reads from a genome FASTA file, for mappability calculations.
- `calculateMappability()`: Generate a BED file defining low mappability regions, using reads generated by `generateSyntheticReads()`, aligned to the genome.

See Also

[Build-Reference-methods](#)

Examples

```
# (1a) Creates genome resource files

ref_path <- file.path(tempdir(), "refWithMapExcl")

getResources(
  reference_path = ref_path,
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)

# (1b) Systematically generate reads based on the example genome:

generateSyntheticReads(
  reference_path = ref_path
)
## Not run:

# (2) Align the generated reads using Rsubread:

# (2a) Build the Rsubread genome index:

subreadIndexPath <- file.path(ref_path, "Rsubread")
if(!dir.exists(subreadIndexPath)) dir.create(subreadIndexPath)
Rsubread::buildindex(
```

```

        basename = file.path(subreadIndexPath, "reference_index"),
        reference = chrZ_genome()
    )

# (2b) Align the synthetic reads using Rsubread::subjunc()

Rsubread::subjunc(
  index = file.path(subreadIndexPath, "reference_index"),
  readfile1 = file.path(ref_path, "Mappability", "Reads.fa"),
  output_file = file.path(ref_path, "Mappability", "AlignedReads.bam"),
  useAnnotation = TRUE,
  annot.ext = chrZ_gtf(),
  isGTF = TRUE
)

# (3) Analyse the aligned reads in the BAM file for low-mappability regions:

calculateMappability(
  reference_path = ref_path,
  aligned_bam = file.path(ref_path, "Mappability", "AlignedReads.bam")
)

# (4) Build the example reference using the calculated Mappability Exclusions

buildRef(ref_path)

# NB the default is to search for the BED file generated by
# `calculateMappability()` in the given reference_path

## End(Not run)

```

NxtSE-class

The NxtSE class

Description

The NxtSE class inherits from the [SummarizedExperiment](#) class and is constructed using [makeSE](#). NxtSE extends SummarizedExperiment by housing additional assays pertaining to IR and splice junction counts.

Usage

```

NxtSE(...)

## S4 method for signature 'NxtSE'
up_inc(x, withDimnames = TRUE, ...)

## S4 method for signature 'NxtSE'
down_inc(x, withDimnames = TRUE, ...)

```

```
## S4 method for signature 'NxtSE'  
up_exc(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
down_exc(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
covfile(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
sampleQC(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
sourcePath(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
row_gr(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
ref(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
junc_PSI(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
junc_counts(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
junc_counts_uns(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
junc_gr(x, withDimnames = TRUE, ...)  
  
## S4 method for signature 'NxtSE'  
update_NxtSE(x, ...)  
  
## S4 method for signature 'NxtSE'  
realize_NxtSE(x, includeJunctions = FALSE, withDimnames = TRUE, ...)  
  
## S4 replacement method for signature 'NxtSE'  
up_inc(x, withDimnames = TRUE) <- value  
  
## S4 replacement method for signature 'NxtSE'  
down_inc(x, withDimnames = TRUE) <- value  
  
## S4 replacement method for signature 'NxtSE'  
up_exc(x, withDimnames = TRUE) <- value
```



```

## S4 replacement method for signature 'NxtSE'
down_exc(x, withDimnames = TRUE) <- value

## S4 replacement method for signature 'NxtSE'
covfile(x, withDimnames = TRUE) <- value

## S4 replacement method for signature 'NxtSE'
sampleQC(x, withDimnames = TRUE) <- value

## S4 method for signature 'NxtSE,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'NxtSE,ANY,ANY,NxtSE'
x[i, j, ...] <- value

## S4 method for signature 'NxtSE'
cbind(..., deparse.level = 1)

## S4 method for signature 'NxtSE'
rbind(..., deparse.level = 1)

```

Arguments

...	In NxtSE(), additional arguments to be passed onto SummarizedExperiment()
x	A NxtSE object
withDimnames	(default TRUE) Whether exported assays should be supplied with row and column names of the NxtSE object. See SummarizedExperiment
includeJunctions	When realizing a NxtSE object, include whether junction counts and PSIs should be realized into memory. Not recommended for general use, as they are only used for coverage plots.
value	The value to replace. Must be a matrix for the up_inc<-, down_inc<-, up_exc<- and down_exc<- replacers, and a character vector for covfile<-
i, j	Row and column subscripts to subset a NxtSE object.
drop	A logical(1), ignored by these methods.
deparse.level	See base::cbind for a description of this argument.

Value

See Functions section (below) for details

Functions

- NxtSE(): Constructor function for NxtSE; akin to SummarizedExperiment(...)
- up_inc(NxtSE): Gets upstream included events (SE/MXE), or upstream exon-intron spanning reads (IR)

- `down_inc(NxtSE)`: Gets downstream included events (SE/MXE), or downstream exon-intron spanning reads (IR)
- `up_exc(NxtSE)`: Gets upstream excluded events (MXE only)
- `down_exc(NxtSE)`: Gets downstream excluded events (MXE only)
- `covfile(NxtSE)`: Gets a named vector with the paths to the corresponding COV files
- `sampleQC(NxtSE)`: Gets a data frame with the QC parameters of the samples
- `sourcePath(NxtSE)`: Retrieves the directory path containing the source data for this NxtSE object.
- `row_gr(NxtSE)`: Retrieves a GRanges object representing the genomic spans of the ASEs (EventRegion as GRanges)
- `ref(NxtSE)`: Retrieves a list of annotation data associated with this NxtSE object; primarily used in `plotCoverage()`
- `junc_PSI(NxtSE)`: Getter for junction PSI DelayedMatrix; primarily used in `plotCoverage()`
- `junc_counts(NxtSE)`: Getter for junction counts DelayedMatrix; primarily used in `plotCoverage()`
- `junc_counts_uns(NxtSE)`: Getter for (unstranded) junction counts DelayedMatrix; primarily used in `plotCoverage()`
- `junc_gr(NxtSE)`: Getter for junction GenomicRanges coordinates; primarily used in `plotCoverage()`
- `update_NxtSE(NxtSE)`: Updates NxtSE object to the latest version.
- `realize_NxtSE(NxtSE)`: Converts all DelayedMatrix assays as matrices (i.e. performs all delayed calculation and loads resulting object to RAM)
- `up_inc(NxtSE) <- value`: Sets upstream included events (SE/MXE), or upstream exon-intron spanning reads (IR)
- `down_inc(NxtSE) <- value`: Sets downstream included events (SE/MXE), or downstream exon-intron spanning reads (IR)
- `up_exc(NxtSE) <- value`: Sets upstream excluded events (MXE only)
- `down_exc(NxtSE) <- value`: Sets downstream excluded events (MXE only)
- `covfile(NxtSE) <- value`: Sets the paths to the corresponding COV files
- `sampleQC(NxtSE) <- value`: Sets the values in the data frame containing sample QC
- `x[i]`: Subsets a NxtSE object
- ``[` (x = NxtSE, i = ANY, j = ANY) <- value`: Sets a subsetted NxtSE object
- `cbind(NxtSE)`: Combines two NxtSE objects (by samples - columns)
- `rbind(NxtSE)`: Combines two NxtSE objects (by AS/IR events - rows)

Examples

```
# Run the full pipeline to generate a NxtSE object:

buildRef(
  reference_path = file.path(tempdir(), "Reference"),
  fasta = chrZ_genome(),
```

```

    gtf = chrZ_gtf()
  )

  bams <- SpliceWiz_example_bams()
  processBAM(bams$path, bams$sample,
    reference_path = file.path(tempdir(), "Reference"),
    output_path = file.path(tempdir(), "SpliceWiz_Output")
  )

  expr <- findSpliceWizOutput(file.path(tempdir(), "SpliceWiz_Output"))
  collateData(expr,
    reference_path = file.path(tempdir(), "Reference"),
    output_path = file.path(tempdir(), "Collated_output")
  )

  se <- makeSE(collate_path = file.path(tempdir(), "Collated_output"))

  # Coerce NxtSE -> SummarizedExperiment
  se_raw <- as(se, "SummarizedExperiment")

  # Coerce SummarizedExperiment -> NxtSE
  se_NxtSE <- as(se_raw, "NxtSE")
  identical(se, se_NxtSE) # Returns TRUE

  # Update NxtSE object to the latest version
  # - useful if an NxtSE object made with old SpliceWiz version
  # - was stored as an RDS object

  se <- update_NxtSE(se)

  # Get directory path of NxtSE (i.e., collate_path)
  sourcePath(se)

  # Get Main Assay Counts
  assay(se, "Included") # Junction (or IR depth) counts for included isoform
  assay(se, "Excluded") # Junction (or IR depth) counts for excluded isoform

  # Get Auxiliary Counts (for filter use only)
  assay(se, "Coverage") # Participation ratio (intron coverage for IR/RI)
  assay(se, "minDepth") # SpliceOver junction counts (Intron Depths for IR/RI)
  assay(se, "Depth")    # Sum of intron depth and SpliceOver (used for
                        # coverage normalization factor

  # Get Junction reads of SE / MXE and spans-reads of IR events
  up_inc(se)  # Upstream included junction counts (IR/MXE/SE/RI)
  down_inc(se) # Downstream included junction counts (IR/MXE/SE/RI)
  up_exc(se)  # Upstream excluded junction counts (MXE only)
  down_exc(se) # Downstream excluded junction counts (MXE only)

  # Get Junction counts
  junc_counts(se) # stranded (if RNA-seq is auto-detected as stranded)
  junc_counts_uns(se) # unstranded (sum of junction reads from both strand)
  junc_PSI(se) # PSI of junction (as proportion of SpliceOver metric)

```

```

# Get Junction GRanges object
junc_gr(se)

# Get EventRegion as GRanges object
row_gr(se)

# Get list of available coverage files
covfile(se)

# Get sample QC information
sampleQC(se)

# Get resource data (used internally for plotCoverage())
cov_data <- ref(se)
names(cov_data)

# Subset functions
se_by_samples <- se[,1:3]
se_by_events <- se[1:10,]
se_by_rowData <- subset(se, EventType == "IR")

# Cbind (bind event_identical NxtSE by samples)
se_by_samples_1 <- se[,1:3]
se_by_samples_2 <- se[,4:6]
se_cbind <- cbind(se_by_samples_1, se_by_samples_2)
identical(se, se_cbind) # should return TRUE

# Rbind (bind sample_identical NxtSE by events)
se_IR <- subset(se, EventType == "IR")
se_SE <- subset(se, EventType == "SE")
se_IRSE <- rbind(se_IR, se_SE)
identical(se_IRSE, subset(se, EventType %in% c("IR", "SE"))) # TRUE

# Convert HDF5-based NxtSE to in-memory se
# makeSE() creates a HDF5-based NxtSE object where all assay data is stored
# as an h5 file instead of in-memory. All operations are performed as
# delayed operations as per DelayedArray package.
# To realize the NxtSE object as an in-memory object, use:

se_real <- realize_NxtSE(se)
identical(se, se_real) # should return FALSE

# To check the difference, run:
class(up_inc(se))
class(up_inc(se_real))

```

Description

Generate plotly / ggplot RNA-seq genome and coverage plots from command line. Note that these are legacy functions. More expansive functionality is available using [getCoverageData](#) / [getPlotObject](#) / [plotView](#) functions.

Usage

```
plotCoverage(  
  se,  
  Event,  
  Gene,  
  seqname,  
  start,  
  end,  
  coordinates,  
  strand = c("*", "+", "-"),  
  zoom_factor = 0.2,  
  bases_flanking = 100,  
  tracks,  
  track_names = tracks,  
  condition,  
  ribbon_mode = c("sd", "ci", "sem", "none"),  
  selected_transcripts = "",  
  reverseGenomeCoords = FALSE,  
  plotJunctions = FALSE,  
  junctionThreshold = 0.01,  
  plot_key_isoforms = FALSE,  
  condense_tracks = FALSE,  
  stack_tracks = FALSE,  
  t_test = FALSE,  
  norm_event,  
  usePlotly = FALSE  
)
```

```
plotGenome(  
  se,  
  reference_path,  
  Event,  
  Gene,  
  seqname,  
  start,  
  end,  
  coordinates,  
  zoom_factor = 0.2,  
  bases_flanking = 100,  
  reverseGenomeCoords = FALSE,  
  condense_tracks = FALSE,  
  selected_transcripts = "",
```

```

    plot_key_isoforms = FALSE,
    usePlotly = FALSE
)

```

Arguments

<code>se</code>	A NxtSE object, created by makeSE . COV files must be linked to the NxtSE object. To do this, see the example in makeSE . Required by <code>plotCoverage</code> . Not required by <code>plotGenome</code> if <code>reference_path</code> is supplied.
<code>Event</code>	The EventName of the IR / alternative splicing event to be displayed. Use <code>rownames(se)</code> to display a list of valid events.
<code>Gene</code>	Whether to use the range for the given Gene. If given, overrides <code>Event</code> (but <code>Event</code> or <code>norm_event</code> will be used to normalise by condition). Valid Gene entries include <code>gene_id</code> (Ensembl ID) or <code>gene_name</code> (Gene Symbol).
<code>seqname, start, end</code>	The chromosome (string) and genomic start/end coordinates (numeric) of the region to display. If present, overrides both <code>Event</code> and <code>Gene</code> . E.g. for a given region of <code>chr1:10000-11000</code> , use the parameters: <code>seqname = "chr1"</code> , <code>start = 10000</code> , <code>end = 11000</code>
<code>coordinates</code>	A string specifying genomic coordinates can be given instead of <code>seqname, start, end</code> . Must be of the format "chr:start-end", e.g. "chr1:10000-11000"
<code>strand</code>	Whether to show coverage of both strands "*" (default), or from the "+" or "-" strand only.
<code>zoom_factor</code>	Zoom out from event. Each level of zoom zooms out by a factor of 3. E.g. for a query region of <code>chr1:10000-11000</code> , if a <code>zoom_factor</code> of 1.0 is given, <code>chr1:99000-12000</code> will be displayed.
<code>bases_flanking</code>	(Default = 100) How many bases flanking the zoomed window. Useful when used in conjunction with <code>zoom_factor == 0</code> . E.g. for a given region of <code>chr1:10000-11000</code> , if <code>zoom_factor = 0</code> and <code>bases_flanking = 100</code> , the region <code>chr1:9900-11100</code> will be displayed.
<code>tracks</code>	The names of individual samples, or the names of the different conditions to be plotted. For the latter, set <code>condition</code> to the specified condition category.
<code>track_names</code>	The names of the tracks to be displayed. If omitted, the <code>track_names</code> will default to the input in <code>tracks</code>
<code>condition</code>	To display normalised coverage per condition, set this to the condition category. If omitted, <code>tracks</code> are assumed to refer to the names of individual samples.
<code>ribbon_mode</code>	(default "sd") Whether coverage ribbons signify standard deviation "sd", 95% confidence interval "ci", standard error of the mean "sem", or none "none". Only applicable when <code>condition</code> is set.
<code>selected_transcripts</code>	(Optional) A vector containing transcript ID or transcript names of transcripts to be displayed on the gene annotation track. Useful to remove minor isoforms that are not relevant to the samples being displayed.
<code>reverseGenomeCoords</code>	(default FALSE) Whether to reverse the genomic coordinates - helpful for intuitive plotting of negative-strand genes

plotJunctions	(default FALSE) If TRUE, sashimi plot junction arcs are plotted. Currently only implemented for plots of individual samples.
junctionThreshold	(default 0.01) The threshold expression of junction reads below which junction arcs will be omitted. This removes cluttering of junction arcs from lowly-expressed (rare) junctions. For individual tracks, this is the fraction of coverage height. For by-condition tracks, this is a PSI threshold.
plot_key_isoforms	(default FALSE) If TRUE, only transcripts involved in the selected Event or pair of Events will be displayed.
condense_tracks	(default FALSE) Whether to collapse the transcript track annotations by gene.
stack_tracks	(default FALSE) Whether to graph all the conditions on a single coverage track. If set to TRUE, each condition will be displayed in a different colour on the same track. Ignored if condition is not set.
t_test	(default FALSE) Whether to perform a pair-wise T-test. Only used if there are TWO condition tracks.
norm_event	Whether to normalise by an event different to that given in "Event". The difference between this and Event is that the genomic coordinates can be centered around a different Event, Gene or region as given in seqname/start/end. If norm_event is different to Event, norm_event will be used for normalisation and Event will be used to define the genomic coordinates of the viewing window. norm_event is required if Event is not set and condition is set.
usePlotly	If TRUE, returns a covPlotly object containing the plotly-based interactive plot. If FALSE, returns a ggplot object.
reference_path	The path of the reference generated by Build-Reference-methods . Required by plotGenome if a NxtSE object is not specified.

Details

In RNA sequencing, alignments to spliced transcripts will "skip" over genomic regions of introns. This can be illustrated in a plot using a horizontal genomic axis, with the vertical axis representing the number of alignments covering each nucleotide. As a result, the coverage "hills" represent the expression of exons, and "valleys" to introns.

Different alternatively-spliced isoforms thus produce different coverage patterns. The change in the coverage across an alternate exon relative to its constitutively-included flanking exons, for example, represents its alternative inclusion or skipping. Similarly, elevation of intron valleys represent increased intron retention.

With multiple replicates per sample, coverage is dependent on library size and gene expression. To compare alternative splicing ratios, normalisation of the coverage of the alternate exon (or alternatively retained intron) relative to their constitutive flanking exons, is required. There is no established method for this normalisation, and can be confounded in situations where flanking elements are themselves alternatively spliced.

SpliceWiz performs this coverage normalisation using the same method as its estimate of spliced / intronic transcript abundance using the SpliceOver method (see details section in [collateData](#)).

This normalisation can be applied to correct for library size and gene expression differences between samples of the same experimental condition. After normalisation, mean and variance of coverage can be computed as ratios relative to total transcript abundance. This method can visualise alternatively included genomic regions including cassette exons, alternate splice site usage, and intron retention.

plotCoverage generates plots showing depth of alignments to the genomic axis. Plots can be generated for individual samples or samples grouped by experimental conditions. In the latter, mean and 95% confidence intervals are shown.

plotGenome generates genome transcript tracks only. Protein-coding regions are denoted by thick rectangles, whereas non-protein coding transcripts or untranslated regions are denoted with thin rectangles. Introns are denoted as lines.

Value

For plotCoverage and plotGenome:

- If usePlotly = FALSE returns a patchwork-assembled static plot
- If usePlotly = TRUE returns a [covPlotly](#) object, which generates a plotly interactive plot when shown using show()

Functions

- plotCoverage(): Legacy function - works by internally calling getCoverageData(), getPlotObject(), then plotView()
- plotGenome(): Legacy function - works by internally calling getGenomeData(), followed by plotAnnoTrack()

Examples

```
se <- SpliceWiz_example_NxtSE(novelSplicing = TRUE)

# Assign annotation of the experimental conditions
colData(se)$treatment <- rep(c("A", "B"), each = 3)

# Verify that the COV files are linked to the NxtSE object:
covfile(se)

# Plot the genome track only, with specified gene:
plotGenome(se, Gene = "SRSF3")

# View the genome track, specifying a genomic region via coordinates:
plotGenome(se, coordinates = "chrZ:10000-20000")

# Return a list of ggplot and plotly objects, also plotting junction counts
plotCoverage(
  se = se,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
  tracks = colnames(se)[1:4], plotJunctions = TRUE
)
```



```

# Plot the same, but as a plotly interactive plot

if(interactive()) {
  p <- plotCoverage(
    se = se,
    Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
    tracks = colnames(se)[1:4], plotJunctions = TRUE,
    usePlotly = TRUE
  )
  show(p)
}

# Plot by condition "treatment", including provisional PSIs
plotCoverage(
  se = se,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
  tracks = c("A", "B"), condition = "treatment", plotJunctions = TRUE
)

# As above, but stack all traces into the same track
# - NB: plotJunctions is disabled when `stack_tracks = TRUE`
plotCoverage(
  se = se,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
  tracks = c("A", "B"), condition = "treatment", stack_tracks = TRUE
)

# Plot the above, but unstacked, and with t-test track
# - NB: plotJunctions is disabled when `stack_tracks = TRUE`
plotCoverage(
  se = se,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
  tracks = c("A", "B"), condition = "treatment", t_test = TRUE
)

# Select only transcripts involved in the selected alternative splicing event
plotCoverage(
  se = se,
  Event = "SE:SRSF3-203-exon4;SRSF3-202-int3",
  tracks = colnames(se)[1:4],
  plot_key_isoforms = TRUE
)

```

processBAM

Runs the OpenMP/C++ based SpliceWiz algorithm

Description

These function calls the SpliceWiz C++ routine on one or more BAM files.

The routine is an improved version over the original IRFinder, with OpenMP-based multi-threading and the production of compact "COV" files to record alignment coverage. A SpliceWiz reference built using [Build-Reference-methods](#) is required.

After processBAM() is run, users should call [collateData](#) to collate individual outputs into an experiment / dataset.

BAM2COV creates COV files from BAM files without running processBAM().

See details for performance info.

Usage

```
BAM2COV(
  bamfiles = "./Unsorted.bam",
  sample_names = "sample1",
  output_path = "./cov_folder",
  n_threads = 1,
  useOpenMP = TRUE,
  overwrite = FALSE,
  verbose = FALSE,
  multiRead = FALSE
)

processBAM(
  bamfiles = "./Unsorted.bam",
  sample_names = "sample1",
  reference_path = "./Reference",
  output_path = "./SpliceWiz_Output",
  n_threads = 1,
  useOpenMP = TRUE,
  overwrite = FALSE,
  run_featureCounts = FALSE,
  verbose = FALSE,
  skipCOVfiles = FALSE,
  multiRead = FALSE
)
```

Arguments

bamfiles	A vector containing file paths of 1 or more BAM files
sample_names	The sample names of the given BAM files. Must be a vector of the same length as bamfiles
output_path	The output directory of this function
n_threads	(default 1) The number of threads to use. See details.
useOpenMP	(default TRUE) Whether to use OpenMP. If set to FALSE, BiocParallel will be used if n_threads is set

overwrite	(default FALSE) If output files already exist, will not attempt to re-run. If run_featureCounts is TRUE, will not overwrite gene counts of previous run unless overwrite is TRUE.
verbose	(default FALSE) Set to TRUE to allow processBAM() to output progress bars and messages
multiRead	(default FALSE) Whether SpliceWiz/ompBAM should use one (set to FALSE) or all available threads (set to TRUE) to read BAM files from the storage drive. In SSD drives or high performance computing clusters, setting to TRUE may slightly improve performance, whereas if reading from disk is the speed bottleneck, the default setting FALSE should result in higher performance.
reference_path	The directory containing the SpliceWiz reference
run_featureCounts	(default FALSE) Whether this function will run Rsubread::featureCounts on the BAM files after counting spliced reads. If so, the output will be saved to "main.FC.Rds" in the output_path directory as a list object.
skipCOVfiles	(default FALSE) Whether processBAM should skip the production of COV files (containing coverage data). Default is to create COV files unless this is set to TRUE. COV files can be generated separately using BAM2COV

Details

Typical run-times for a 100-million paired-end alignment BAM file takes 10 minutes using a single core. Using 8 threads, the runtime is approximately 2-5 minutes, depending on your system's file input / output speeds. Approximately 10 Gb of RAM is used when OpenMP is used. If OpenMP is not used (see below), this memory usage is multiplied across the number of processor threads (i.e. 40 Gb if n_threads = 4).

OpenMP is natively available to Linux / Windows compilers, and OpenMP will be used if useOpenMP is set to TRUE, using multiple threads to process each BAM file. On Macs, if OpenMP is not available at compilation, BiocParallel will be used, processing BAM files simultaneously, with one BAM file per thread.

Value

Output will be saved to output_path. Output files will be named using the given sample_names. For processBAM():

- sample.txt.gz: The main output file containing the quantitation of IR and splice junctions, as well as QC information
- sample.cov: Contains coverage information in compressed binary. See [getCoverage](#)
- main.FC.Rds: A single file containing gene counts for the whole dataset (only if run_featureCounts == TRUE)

For BAM2COV():

- sample.cov: Contains coverage information in compressed binary. See [getCoverage](#)

Functions

- BAM2COV(): Converts BAM files to COV files without running processBAM()
- processBAM(): Processes BAM files. Requires a SpliceWiz reference generated by buildRef()

See Also

[Build-Reference-methods collateData isCOV](#)

Examples

```
# Run BAM2COV, which only produces COV files but does not run `processBAM()`:

bams <- SpliceWiz_example_bams()

BAM2COV(bams$path, bams$sample,
  output_path = file.path(tempdir(), "SpliceWiz_Output"),
  n_threads = 2, overwrite = TRUE
)

# Run processBAM(), which produces:
# - text output of intron coverage and spliced read counts
# - COV files which record read coverages

example_ref <- file.path(tempdir(), "Reference")

buildRef(
  reference_path = example_ref,
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)

bams <- SpliceWiz_example_bams()

processBAM(bams$path, bams$sample,
  reference_path = file.path(tempdir(), "Reference"),
  output_path = file.path(tempdir(), "SpliceWiz_Output"),
  n_threads = 2
)
```

Run_SpliceWiz_Filters *Filtering for IR and Alternative Splicing Events*

Description

This function implements filtering of alternative splicing events, based on customisable criteria. See [ASEFilter](#) for details on how to construct SpliceWiz filters

Usage

```

getDefaultFilters()

applyFilters(se, filters = getDefaultFilters())

runFilter(se, filterObj)

```

Arguments

se	the NxtSE object to filter
filters	A vector or list of one or more ASEFilter objects. If left blank, the SpliceWiz default filters will be used.
filterObj	A single ASEFilter object.

Details

We highly recommend using the default filters, which are as follows:

- (1) Depth filter of 20,
- (2) Participation filter requiring 70% coverage in IR events.
- (3) Participation filter requiring 40% coverage in MXE, SE, A5SS and A3SS events (i.e. Included + Excluded isoforms must cover at least 40% of all junction events across the given region)
- (4) Consistency filter requiring log difference of 2 (for skipped exon and mutually exclusive exon events, each junction must comprise at least $1/(2^2) = 1/4$ of all reads associated with each isoform). For retained introns, the exon-intron overhangs must not differ by 1/4
- (5) Terminus filter: In alternate first exons, the splice junction must not be shared with another transcript for which it is not its first intron. For alternative last exons, the splice junction must not be shared with another transcript for which it is not its last intron
- (6) ExclusiveMXE filter: For MXE events, the two alternate cassette exons must not overlap in their genomic regions
- (7) StrictAltSS filter: For A5SS / A3SS events, the two alternate splice sites must not be interrupted by an intron

In all data-based filters, we require at least 80% samples ($pcTRUE = 80$) to pass this filters from the entire dataset ($minCond = -1$).

Threshold depths for Participation filters:

For IR/RI, Participation filter is only applied for IR events for which the intron depth is above a certain threshold (set by $minDepth$). This avoids the filters running on samples for which there is no IR.

For non-IR ASEs, Participation is only run on events with splice depth ($SpliceOver$ metric) higher than $minDepth$. This avoids filters running on events with low total participation (i.e., $(Inc+Exc)/SpliceOver$)

Threshold depths for Consistency filters: Consistency filters are only applied for events where the sum of upstream and downstream junction counts surpass a given threshold $minDepth$. This is

applied on both included and excluded counts (the latter only applies to MXE). This avoids consistency filters running on events with insufficient junction counts (leading to high variance between up/downstream values).

For an explanation of the various parameters mentioned here, see [ASEFilter](#)

Value

For `runFilter` and `applyFilters`: a vector of type `logical`, representing the rows of `NxtSE` that should be kept.

For `getDefaultFilters`: returns a list of default recommended filters that should be parsed into `applyFilters`.

Functions

- `getDefaultFilters()`: Returns a vector of recommended default SpliceWiz filters
- `applyFilters()`: Run a vector or list of `ASEFilter` objects on a `NxtSE` object
- `runFilter()`: Run a single filter on a `NxtSE` object

See Also

[ASEFilter](#) for details describing how to create and assign settings to `ASEFilter` objects.

Examples

```
# see ?makeSE on example code of how this object was generated
se <- SpliceWiz_example_NxtSE()

# Get the list of SpliceWiz recommended filters
filters <- getDefaultFilters()

# View a description of what these filters do:
filters

# Filter the NxtSE using the first default filter ("Depth")
se.depthfilter <- se[runFilter(se, filters[[1]]), ]

# Filter the NxtSE using all four default filters
se.defaultFiltered <- se[applyFilters(se, getDefaultFilters()), ]
```

setSWthreads	<i>Sets the number of threads used by SpliceWiz</i>
--------------	---

Description

SpliceWiz uses the computationally efficient packages `fst` and `data.table` to compute file and data operations, respectively. Both packages make use of parallelisation. If excessive number of threads are allocated, it may impact the running of other operations on your system. Use this function to manually allocate the desired number of threads

Usage

```
setSWthreads(threads = 0)
```

Arguments

threads	(default 0) The number of threads for SpliceWiz to use. Set as 0 to use the recommended number of threads appropriate for the system (approximately half the available threads)
---------	---

Value

Nothing.

Examples

```
setSWthreads(0)
```

STAR-methods	<i>STAR wrappers for building reference for STAR, and aligning RNA-sequencing</i>
--------------	---

Description

These STAR helper / wrapper functions allow users to (1) create a STAR genome reference (with or without GTF), (2) align one or more RNA-seq samples, and (3) calculate regions of low mappability. STAR references can be created using one-step (genome and GTF), or two-step (genome first, then on-the-fly with injected GTF) approaches.

Usage

```
STAR_version()

STAR_buildRef(
  reference_path,
  STAR_ref_path = file.path(reference_path, "STAR"),
  n_threads = 4,
  overwrite = FALSE,
  sjdbOverhang = 100,
  sparsity = 1,
  also_generate_mappability = FALSE,
  map_depth_threshold = 4,
  additional_args = NULL,
  ...
)

STAR_alignExperiment(
  Experiment,
  STAR_ref_path,
  BAM_output_path,
  n_threads = 4,
  overwrite = FALSE,
  two_pass = FALSE,
  trim_adaptor = "AGATCGGAAG",
  additional_args = NULL
)

STAR_alignReads(
  fastq_1 = c("./sample_1.fastq"),
  fastq_2 = NULL,
  STAR_ref_path,
  BAM_output_path,
  n_threads = 4,
  overwrite = FALSE,
  two_pass = FALSE,
  trim_adaptor = "AGATCGGAAG",
  memory_mode = "NoSharedMemory",
  additional_args = NULL
)

STAR_buildGenome(
  reference_path,
  STAR_ref_path = file.path(reference_path, "STAR"),
  n_threads = 4,
  overwrite = FALSE,
  sparsity = 1,
  also_generate_mappability = FALSE,
  map_depth_threshold = 4,
```



```

    additional_args = NULL,
    ...
)

STAR_loadGenomeGTF(
  reference_path,
  STAR_ref_path,
  STARgenome_output = file.path(tempdir(), "STAR"),
  n_threads = 4,
  overwrite = FALSE,
  sjdbOverhang = 100,
  extraFASTA = "",
  additional_args = NULL
)

STAR_mappability(
  reference_path,
  STAR_ref_path = file.path(reference_path, "STAR"),
  map_depth_threshold = 4,
  n_threads = 4,
  ...
)

```

Arguments

<code>reference_path</code>	The path to the reference. getResources must first be run using this path as its <code>reference_path</code>
<code>STAR_ref_path</code>	(Default - the "STAR" subdirectory under <code>reference_path</code>) The directory containing the STAR reference to be used or to contain the newly-generated STAR reference
<code>n_threads</code>	The number of threads to run the STAR aligner.
<code>overwrite</code>	(default FALSE) For <code>STAR_buildRef</code> , <code>STAR_buildGenome</code> and <code>STAR_loadGenomeGTF</code> - if STAR genome already exists, should it be overwritten. For <code>STAR_alignExperiment</code> and <code>STAR_alignReads</code> - if BAM file already exists, should it be overwritten.
<code>sjdbOverhang</code>	(Default = 100) A STAR setting indicating the length of the donor / acceptor sequence on each side of the junctions. Ideally equal to $(\text{mate_length} - 1)$. See the STAR aligner manual for details.
<code>sparsity</code>	(default 1) Sets STAR's <code>--genomeSAsparseD</code> option. For human (and mouse) genomes, set this to 2 to allow STAR to perform genome generation and mapping using < 16 Gb of RAM, albeit with slightly lower mapping rate (~ 0.1% lower, according to STAR's author). Setting this to higher values is experimental (and not tested)
<code>also_generate_mappability</code>	Whether <code>STAR_buildRef()</code> and <code>STAR_buildGenome()</code> also calculate Mappability Exclusion regions.
<code>map_depth_threshold</code>	(Default 4) The depth of mapped reads threshold at or below which Mappability

	exclusion regions are defined. See Mappability-methods . Ignored if <code>also_generate_mappability = FALSE</code>
<code>additional_args</code>	A character vector of additional arguments to be parsed into STAR. See examples below.
<code>...</code>	Additional arguments to be parsed into <code>generateSyntheticReads()</code> . See Mappability-methods .
<code>Experiment</code>	A two or three-column data frame with the columns denoting sample names, forward-FASTQ and reverse-FASTQ files. This can be conveniently generated using findFASTQ
<code>BAM_output_path</code>	The path under which STAR outputs the aligned BAM files. In <code>STAR_alignExperiment()</code> , STAR will output aligned BAMS inside subdirectories of this folder, named by sample names. In <code>STAR_alignReads()</code> , STAR will output directly into this path.
<code>two_pass</code>	Whether to use two-pass mapping. In <code>STAR_alignExperiment()</code> , STAR first-pass will align every sample to generate a list of splice junctions but not BAM files. The junctions are then given to STAR to generate a temporary genome containing information about novel junctions, thereby improving novel junction detection. In <code>STAR_alignReads()</code> , STAR will use <code>--twopassMode Basic</code>
<code>trim_adaptor</code>	The sequence of the Illumina adaptor to trim via STAR's <code>--clip3pAdapterSeq</code> option
<code>fastq_1, fastq_2</code>	In <code>STAR_alignReads</code> : character vectors giving the path(s) of one or more FASTQ (or FASTA) files to be aligned. If single reads are to be aligned, omit <code>fastq_2</code>
<code>memory_mode</code>	The parameter to be parsed to <code>--genomeLoad</code> ; either <code>NoSharedMemory</code> or <code>LoadAndKeep</code> are used.
<code>STARgenome_output</code>	The output path of the created on-the-fly genome
<code>extraFASTA</code>	(default <code>""</code>) One or more FASTA files containing spike-in genome sequences (e.g. ERCC, Sequins), as required.

Details

Pre-requisites

`STAR_buildRef()` and `STAR_buildGenome()` require prepared genome and gene annotation reference retrieved using [getResources](#), which is run internally by [buildRef](#)

`STAR_loadGenomeGTF()` requires the above, and additionally a STAR genome created using `STAR_buildGenome()`

`STAR_alignExperiment()`, `STAR_alignReads()`, and `STAR_mappability()`: requires a STAR genome, which can be built using `STAR_buildRef()` or `STAR_buildGenome()` followed by `STAR_loadGenomeGTF()`

Function Description

For `STAR_buildRef`: this function will create a STAR genome reference using the same genome FASTA and gene annotation GTF used to create the SpliceWiz reference. Optionally, it will run `STAR_mappability` if `also_generate_mappability` is set to `TRUE`

For `STAR_alignExperiment`: aligns a set of FASTQ or paired FASTQ files using the given STAR genome using the STAR aligner. A data.frame specifying sample names and corresponding FASTQ files are required

For `STAR_alignReads`: aligns a single or pair of FASTQ files to the given STAR genome using the STAR aligner.

For `STAR_buildGenome`: Creates a STAR genome reference, using ONLY the FASTA file used to create the SpliceWiz reference. This allows users to create a single STAR reference for use with multiple transcriptome (GTF) references (on different occasions). Optionally, it will run `STAR_mappability` if `also_generate_mappability` is set to TRUE

For `STAR_loadGenomeGTF`: Creates an "on-the-fly" STAR genome, injecting GTF from the given SpliceWiz `reference_path`, setting `sjdbOverhang` setting, and (optionally) any spike-ins via the `extraFASTA` parameter. This allows users to create a single STAR reference for use with multiple transcriptome (GTF) references, with different `sjdbOverhang` settings, and/or spike-ins (on different occasions or for different projects).

For `STAR_mappability`: this function will first will run [generateSyntheticReads](#), then use the given STAR genome to align the synthetic reads using STAR. The aligned BAM file will then be processed using [calculateMappability](#) to calculate the lowly-mappable genomic regions, producing the `MappabilityExclusion.bed.gz` output file.

Value

For `STAR_version()`: The STAR version

For `STAR_buildRef()`: None

For `STAR_alignExperiment()`: None

For `STAR_alignReads()`: None

For `STAR_buildGenome()`: None

For `STAR_loadGenomeGTF()`: The path of the on-the-fly STAR genome, typically in the subdirectory `"_STARgenome"` within the given `STARgenome_output` directory

For `STAR_mappability()`: None

Functions

- `STAR_version()`: Checks whether STAR is installed, and its version
- `STAR_buildRef()`: Creates a STAR genome reference, using both FASTA and GTF files used to create the SpliceWiz reference
- `STAR_alignExperiment()`: Aligns multiple sets of FASTQ files, belonging to multiple samples
- `STAR_alignReads()`: Aligns a single sample (with single or paired FASTQ or FASTA files)
- `STAR_buildGenome()`: Creates a STAR genome reference, using ONLY the FASTA file used to create the SpliceWiz reference
- `STAR_loadGenomeGTF()`: Creates an "on-the-fly" STAR genome, injecting GTF from the given SpliceWiz `reference_path`, setting `sjdbOverhang` setting, and (optionally) any spike-ins as `extraFASTA`
- `STAR_mappability()`: Calculates lowly-mappable genomic regions using STAR

See Also

[Build-Reference-methods findSamples Mappability-methods](#)

[The latest STAR documentation](#)

Examples

```
# 0) Check that STAR is installed and compatible with SpliceWiz

STAR_version()
## Not run:

# The below workflow illustrates
# 1) Getting the reference resource
# 2) Building the STAR Reference, including Mappability Exclusion calculation
# 3) Building the SpliceWiz Reference, using the Mappability Exclusion file
# 4) Aligning (a) one or (b) multiple raw sequencing samples.

# 1) Reference generation from Ensembl's FTP links

FTP <- "ftp://ftp.ensembl.org/pub/release-94/"

getResources(
  reference_path = "Reference_FTP",
  fasta = paste0(FTP, "fasta/homo_sapiens/dna/",
    "Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz"),
  gtf = paste0(FTP, "gtf/homo_sapiens/",
    "Homo_sapiens.GRCh38.94.chr.gtf.gz")
)

# 2) Generates STAR genome within the SpliceWiz reference. Also generates
# mappability exclusion gzipped BED file inside the "Mappability/" sub-folder

STAR_buildRef(
  reference_path = "Reference_FTP",
  STAR_ref_path = file.path("Reference_FTP", "STAR"),
  n_threads = 8,
  also_generate_mappability = TRUE
)

# 2a) Generates STAR genome of the example SpliceWiz genome.
# This demonstrates using custom STAR parameters, as the example
# SpliceWiz genome is ~100k in length,
# so --genomeSAindexNbases needs to be
# adjusted to be min(14, log2(GenomeLength)/2 - 1)

getResources(
  reference_path = "Reference_chrZ",
  fasta = chrZ_genome(),
  gtf = chrZ_gtf()
)
```

```
STAR_buildRef(  
  reference_path = "Reference_chrZ",  
  STAR_ref_path = file.path("Reference_chrZ", "STAR"),  
  n_threads = 8,  
  additional_args = c("--genomeSAindexNbases", "7"),  
  also_generate_mappability = TRUE  
)  
  
# 3) Build SpliceWiz reference using the newly-generated  
#   Mappability exclusions  
  
#' NB: also specifies to use the hg38 nonPolyA resource  
  
buildRef(reference_path = "Reference_FTP", genome_type = "hg38")  
  
# 4a) Align a single sample using the STAR reference  
  
STAR_alignReads(  
  fastq_1 = "sample1_1.fastq", fastq_2 = "sample1_2.fastq",  
  STAR_ref_path = file.path("Reference_FTP", "STAR"),  
  BAM_output_path = "./bams/sample1",  
  n_threads = 8  
)  
  
# 4b) Align multiple samples, using two-pass alignment  
  
Experiment <- data.frame(  
  sample = c("sample_A", "sample_B"),  
  forward = file.path("raw_data", c("sample_A", "sample_B"),  
    c("sample_A_1.fastq", "sample_B_1.fastq")),  
  reverse = file.path("raw_data", c("sample_A", "sample_B"),  
    c("sample_A_2.fastq", "sample_B_2.fastq"))  
)  
  
STAR_alignExperiment(  
  Experiment = Experiment,  
  STAR_ref_path = file.path("Reference_FTP", "STAR"),  
  BAM_output_path = "./bams",  
  n_threads = 8,  
  two_pass = TRUE  
)  
  
# - Building a STAR genome (only) reference, and injecting GTF as a  
#   subsequent step  
#  
#   This is useful for users who want to create a single STAR genome, for  
#   experimentation with different GTF files.  
#   It is important to note that the chromosome names of the genome (FASTA)  
#   file and the GTF file needs to be identical. Thus, Ensembl and Gencode  
#   GTF files should not be mixed (unless the chromosome GTF names have  
#   been fixed)
```

```

# - also set sparsity = 2 to build human genome so that it will fit in
# 16 Gb RAM. NB: this step's RAM usage can be set using the
# `--limitGenomeGenerateRAM` parameter

STAR_buildGenome(
  reference_path = "Reference_FTP",
  STAR_ref_path = file.path("Reference_FTP", "STAR_genomeOnly"),
  n_threads = 8, sparsity = 2,
  additional_args = c("--limitGenomeGenerateRAM", "16000000000")
)

# - Injecting a GTF into a genome-only STAR reference
#
# This creates an on-the-fly STAR genome, using a GTF file
# (derived from a SpliceWiz reference) into a new location.
# This allows a single STAR reference to use multiple GTFs
# on different occasions.

STAR_new_ref <- STAR_loadGenomeGTF(
  reference_path = "Reference_FTP",
  STAR_ref_path = file.path("Reference_FTP", "STAR_genomeOnly"),
  STARgenome_output = file.path(tempdir(), "STAR"),
  n_threads = 4,
  sjdbOverhang = 100
)

# This new reference can then be used to align your experiment:

STAR_alignExperiment(
  Experiment = Experiment,
  STAR_ref_path = STAR_new_ref,
  BAM_output_path = "./bams",
  n_threads = 8,
  two_pass = TRUE
)

# Typically, one should `clean up` the on-the-fly STAR reference (as it is
# large!). If it is in a temporary directory, it will be cleaned up
# when the current R session ends; otherwise this needs to be done manually:

unlink(file.path(tempdir(), "STAR"), recursive = TRUE)

## End(Not run)

```

 theme_white

ggplot2 themes

Description

A ggplot theme object for white background figures +/- a legend

Usage

```
theme_white  
  
theme_white_legend  
  
theme_white_legend_plot_track
```

Format

An object of class theme (inherits from gg) of length 10.

An object of class theme (inherits from gg) of length 9.

An object of class theme (inherits from gg) of length 10.

Functions

- `theme_white`: White theme without figure legend
- `theme_white_legend`: White theme but with a figure legend (if applicable)
- `theme_white_legend_plot_track`: White theme with figure legend but without horizontal grid lines. Used internally in PlotGenome

See Also

[plotCoverage](#)

Examples

```
library(ggplot2)  
df <- data.frame(  
  gp = factor(rep(letters[1:3], each = 10)),  
  y = rnorm(30))  
ggplot(df, aes(gp, y)) +  
  geom_point() +  
  theme_white
```

View-Reference-methods

View SpliceWiz Reference in read-able data frames

Description

These functions allow users to construct tables containing SpliceWiz's reference of alternate splicing events, intron retention events, and other relevant data

Usage

```
viewASE(reference_path)

viewIR(reference_path, directional = TRUE)

viewIntrons(reference_path)

viewIR_NMD(reference_path)

viewExons(reference_path)

viewGenes(reference_path)

viewGO(reference_path)

viewProteins(reference_path)

viewTranscripts(reference_path)
```

Arguments

`reference_path` The directory containing the SpliceWiz reference

`directional` (default TRUE) Whether to view IR events for stranded RNAseq TRUE or un-stranded protocol FALSE

Value

A data frame containing the relevant info. See details

Functions

- `viewASE()`: Outputs summary of alternative splicing events constructed by SpliceWiz
- `viewIR()`: Outputs summary of assessed IRFInde-like IR events, constructed by SpliceWiz
- `viewIntrons()`: Outputs summary of all introns from the annotation, constructed by SpliceWiz
- `viewIR_NMD()`: Outputs information for every intron - whether retention of the intron will convert the transcript to an NMD substrate
- `viewExons()`: Outputs information for every exon from the annotation.
- `viewGenes()`: Outputs information for every gene from the annotation.
- `viewGO()`: Outputs information for every gene from the annotation.
- `viewProteins()`: Outputs information for every protein-coding exon from the annotation.
- `viewTranscripts()`: Outputs information for every transcript from the annotation.

See Also

[Build-Reference-methods](#)

Examples

```
ref_path <- file.path(tempdir(), "Reference_withGO")
buildRef(
  reference_path = ref_path,
  fasta = chrZ_genome(),
  gtf = chrZ_gtf(),
  ontologySpecies = "Homo sapiens"
)

df <- viewASE(ref_path)

df <- viewIR(ref_path, directional = TRUE)

df <- viewIntrons(ref_path)

df <- viewIR_NMD(ref_path)

df <- viewExons(ref_path)

df <- viewGenes(ref_path)

df <- viewProteins(ref_path)

df <- viewTranscripts(ref_path)

df <- viewGO(ref_path)
```

Index

- * **datasets**
 - theme_white, 86
- * **package**
 - example-SpliceWiz-data, 46
 - SpliceWiz-package, 3
- [,NxtSE,ANY,ANY,ANY-method (NxtSE-class), 63
- [<-,NxtSE,ANY,ANY,NxtSE-method (NxtSE-class), 63

- addPSI_edgeR (ASE-GLM-edgeR), 5
- AnnotationHub, 25
- applyFilters, 4, 6, 12
- applyFilters (Run_SpliceWiz_Filters), 76
- ASE-GLM-edgeR, 5, 13
- ASE-methods, 4, 7, 10, 14, 56, 57, 59
- ASE_DESeq (ASE-methods), 10
- ASE_DoubleExpSeq (ASE-methods), 10
- ASE_edgeR (ASE-methods), 10
- ASE_edgeR_timeseries (ASE-methods), 10
- ASE_limma (ASE-methods), 10
- ASE_limma_timeseries (ASE-methods), 10
- ASEFilter, 4, 76–78
- ASEFilter (ASEFilter-class), 17
- ASEFilter-class, 17

- BAM2COV, 35, 75
- BAM2COV (processBAM), 73
- base::cbind, 65
- Build-Reference-methods, 4, 20, 28, 33, 52, 62, 71, 74, 76, 84, 88
- buildFullRef, 62
- buildFullRef (Build-Reference-methods), 20
- buildRef, 4, 62, 82
- buildRef (Build-Reference-methods), 20

- calculateMappability, 83
- calculateMappability (Mappability-methods), 60

- calculateMappability(), 22
- cbind,NxtSE-method (NxtSE-class), 63
- coerce, SummarizedExperiment, NxtSE-method (NxtSE-class), 63
- colData, 4, 56
- collateData, 4, 6, 12, 14, 28, 30, 49, 56, 57, 71, 74, 76
- condition (covPlotObject-class), 40
- condition, covPlotObject-method (covPlotObject-class), 40
- coord2GR, 31
- covDataObject-class, 32
- Coverage, 4, 35
- covfile (NxtSE-class), 63
- covfile, NxtSE-method (NxtSE-class), 63
- covfile<- (NxtSE-class), 63
- covfile<-, NxtSE-method (NxtSE-class), 63
- covPlotly, 44, 72
- covPlotly-class, 38
- covPlotObject, 34
- covPlotObject-class, 40

- DESeq2::results, 15
- DoubleExpSeq::DBGLM1, 15
- down_exc (NxtSE-class), 63
- down_exc, NxtSE-method (NxtSE-class), 63
- down_exc<- (NxtSE-class), 63
- down_exc<-, NxtSE-method (NxtSE-class), 63
- down_inc (NxtSE-class), 63
- down_inc, NxtSE-method (NxtSE-class), 63
- down_inc<- (NxtSE-class), 63
- down_inc<-, NxtSE-method (NxtSE-class), 63

- edgeR::topTags, 15
- example-SpliceWiz-data, 46
- extract_gene_ids_for_GO (Gene-ontology-methods), 50

- findBAMS (findSamples), 48
- findFASTQ, 82
- findFASTQ (findSamples), 48
- findSamples, 28, 48, 84
- findSpliceWizOutput, 28
- findSpliceWizOutput (findSamples), 48
- fitASE_edgeR (ASE-GLM-edgeR), 5
- fitASE_edgeR_custom (ASE-GLM-edgeR), 5

- Gene-ontology-methods, 50
- generateSyntheticReads, 83
- generateSyntheticReads (Mappability-methods), 60
- GenomicRanges::findOverlaps, 30
- getAvailableGO (Build-Reference-methods), 20
- getCoverage, 75
- getCoverage (Coverage), 35
- getCoverage_DF (Coverage), 35
- getCoverageBins (Coverage), 35
- getCoverageData, 5, 44, 69
- getCoverageData (covDataObject-class), 32
- getCoverageRegions (Coverage), 35
- getDefaultFilters, 19
- getDefaultFilters (Run_SpliceWiz_Filters), 76
- getExonRanges (covPlotly-class), 38
- getExonRanges, covPlotly-method (covPlotly-class), 38
- getGenomeData (covDataObject-class), 32
- getNonPolyARef (Build-Reference-methods), 20
- getPlotObject, 5, 69
- getPlotObject (covPlotObject-class), 40
- getResources, 61, 81, 82
- getResources (Build-Reference-methods), 20
- goASE (Gene-ontology-methods), 50
- goGenes (Gene-ontology-methods), 50
- Graphics-User-Interface, 54
- GUI (Graphics-User-Interface), 54

- isCOV, 55, 76

- junc_counts (NxtSE-class), 63
- junc_counts, NxtSE-method (NxtSE-class), 63
- junc_counts_uns (NxtSE-class), 63

- junc_counts_uns, NxtSE-method (NxtSE-class), 63
- junc_gr (NxtSE-class), 63
- junc_gr, NxtSE-method (NxtSE-class), 63
- junc_PSI (NxtSE-class), 63
- junc_PSI, NxtSE-method (NxtSE-class), 63

- limma::topTable, 15

- make_plot_data, 4, 56, 58
- makeMatrix, 7, 14, 59
- makeMatrix (make_plot_data), 58
- makeMeanPSI, 7, 14
- makeMeanPSI (make_plot_data), 58
- makeSE, 4, 29, 30, 33, 46, 47, 56, 59, 63, 70
- Mappability-methods, 24, 25, 60, 82, 84

- NxtIRFdata::example_bams, 46
- NxtSE, 4, 6, 12, 33, 46, 47, 56, 57, 59, 70, 71, 77
- NxtSE (NxtSE-class), 63
- NxtSE-class, 63
- NxtSE-methods (NxtSE-class), 63

- ompBAM::ompBAM-package, 3

- plotAnnoTrack (covDataObject-class), 32
- plotCoverage, 7, 14, 56, 57, 68, 87
- plotGenome (plotCoverage), 68
- plotGO (Gene-ontology-methods), 50
- plotView, 5, 38, 39, 69
- plotView (covPlotObject-class), 40
- processBAM, 3, 4, 22, 24, 28–30, 35, 48, 49, 56, 73

- rbind, NxtSE-method (NxtSE-class), 63
- realize_NxtSE, 57
- realize_NxtSE (NxtSE-class), 63
- realize_NxtSE, NxtSE-method (NxtSE-class), 63
- ref (NxtSE-class), 63
- ref, NxtSE-method (NxtSE-class), 63
- row_gr (NxtSE-class), 63
- row_gr, NxtSE-method (NxtSE-class), 63
- rowData, 4
- rowMeans, 59
- Rsubread::featureCounts, 75
- Run_SpliceWiz_Filters, 19, 76
- runFilter (Run_SpliceWiz_Filters), 76

- sampleQC (NxtSE-class), 63
- sampleQC, NxtSE-method (NxtSE-class), 63
- sampleQC<- (NxtSE-class), 63
- sampleQC<-, NxtSE-method (NxtSE-class), 63
- setResolution, 43
- setResolution (covPlotly-class), 38
- setResolution, covPlotly-method (covPlotly-class), 38
- setSWthreads, 79
- showEvents (covDataObject-class), 32
- showEvents, covDataObject-method (covDataObject-class), 32
- showExons (covPlotly-class), 38
- showExons, covPlotly-method (covPlotly-class), 38
- sourcePath (NxtSE-class), 63
- sourcePath, NxtSE-method (NxtSE-class), 63
- spliceWiz (Graphics-User-Interface), 54
- SpliceWiz-package, 3
- SpliceWiz_example_bams (example-SpliceWiz-data), 46
- SpliceWiz_example_NxtSE (example-SpliceWiz-data), 46
- STAR-methods, 4, 23, 25, 79
- STAR_alignExperiment (STAR-methods), 79
- STAR_alignReads (STAR-methods), 79
- STAR_buildGenome (STAR-methods), 79
- STAR_buildRef, 23
- STAR_buildRef (STAR-methods), 79
- STAR_loadGenomeGTF (STAR-methods), 79
- STAR_mappability, 22, 61
- STAR_mappability (STAR-methods), 79
- STAR_version (STAR-methods), 79
- stats::plogis, 59
- stats::qlogis, 59
- subset_EventNames_by_GO (Gene-ontology-methods), 50
- SummarizedExperiment, 4, 63, 65

- testASE_edgeR (ASE-GLM-edgeR), 5
- theme_white, 86
- theme_white_legend (theme_white), 86
- theme_white_legend_plot_track (theme_white), 86
- tracks (covPlotObject-class), 40
- tracks, covPlotObject-method (covPlotObject-class), 40

- up_exc (NxtSE-class), 63
- up_exc, NxtSE-method (NxtSE-class), 63
- up_exc<- (NxtSE-class), 63
- up_exc<-, NxtSE-method (NxtSE-class), 63
- up_inc (NxtSE-class), 63
- up_inc, NxtSE-method (NxtSE-class), 63
- up_inc<- (NxtSE-class), 63
- up_inc<-, NxtSE-method (NxtSE-class), 63
- update_NxtSE (NxtSE-class), 63
- update_NxtSE, NxtSE-method (NxtSE-class), 63

- View-Reference-methods, 87
- viewASE (View-Reference-methods), 87
- viewExons (View-Reference-methods), 87
- viewGenes, 51
- viewGenes (View-Reference-methods), 87
- viewGO (View-Reference-methods), 87
- viewIntrons (View-Reference-methods), 87
- viewIR (View-Reference-methods), 87
- viewIR_NMD (View-Reference-methods), 87
- viewProteins (View-Reference-methods), 87
- viewTranscripts (View-Reference-methods), 87