Package 'extraChIPs'

November 12, 2026

Version 1.16.3

Title Additional functions for working with ChIP-Seq data

Description This package builds on existing tools and adds some simple but extremely useful capabilities for working wth ChIP-Seq data. The focus is on detecting differential binding windows/regions.
One set of functions focusses on set-operations retaining mcols for GRanges objects, whilst another group of functions are to aid visualisation of results.
Coercion to tibble objects is also implemented.

License GPL-3

Encoding UTF-8

URL https://github.com/smped/extraChIPs

BugReports https://github.com/smped/extraChIPs/issues

Depends BiocParallel, R (>= 4.2.0), GenomicRanges, ggplot2 (>= 4.0.0), ggside (>= 0.4.0), Seqinfo, SummarizedExperiment (>= 1.39.1), tibble

Imports csaw, dplyr (>= 1.1.1), edgeR (>= 4.0), forcats, GenomeInfoDb, glue, ggrepel, InteractionSet, IRanges, matrixStats, methods, patchwork, RColorBrewer, rlang, Rsamtools, rtracklayer, S4Vectors, scales, stats, stringr, tidyr, tidyselect, vctrs

Suggests apeglm, BiocStyle, SimpleUpset, covr, DESeq2, EnrichedHeatmap, GenomicAlignments, GenomicInteractions, Gviz, ggforce, harmonicmeanp, here, knitr, limma, magrittr, plyranges, quantro, rmarkdown, testthat (>= 3.0.0), tidyverse, VennDiagram

biocViews ChIPSeq, HiC, Sequencing, Coverage

BiocType Software

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

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

git_branch RELEASE_3_23

git_last_commit ebb04b2

git_last_commit_date 2026-10-08

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Stevie Pederson [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8197-3303>)

Maintainer Stevie Pederson <stephen.pederson.au@gmail.com>

Contents

extraChIPs-package . . . 3
.makeFinalProfileHeatmap . . . 4
.mapFeatures . . . 5
.mapGi . . . 6
.mapWithin . . . 7
addDiffStatus . . . 7
as_tibble . . . 9
bestOverlap . . . 10
centrePeaks . . . 12
chopMC . . . 13
collapseGenes . . . 14
colToRanges . . . 15
cytobands . . . 16
defineRegions . . . 17
defineSeqinfo . . . 19
distinctMC . . . 20
dualFilter . . . 21
ex_datasets . . . 23
fitAssayDiff . . . 24
fixed_width_datasets . . . 26
getProfileData . . . 27
grlToSE . . . 29
importPeaks . . . 31
makeConsensus . . . 32
mapByFeature . . . 33
mapGrlCols . . . 36
mergeByCol . . . 37
mergeByHMP . . . 39
mergeBySig . . . 41
partitionRanges . . . 43
plotAssayDensities . . . 45
plotAssayHeatmap . . . 46
plotAssayPCA . . . 48

plotAssayRle 49
plotGrlCol 51
plotHFGC 53
plotOverlaps 59
plotPairwise 61
plotPie 64
plotProfileHeatmap 67
plotSplitDonut 70
propOverlap 74
reduceMC 75
setoptsMC 76
stitchRanges 77
voomWeightsFromCPM 78

Index **80**

`extraChIPs-package` *extraChIPs: A package for enabling and extending ChIP-Seq analysis*

Description

The package provides three categories of important functions: Range-based, Visualisation and Convenience functions, with most centred around GenomicRanges objects

Range-based Functions

Many of the range-based functions included in this package have a focus on retaining the `mcols` information whilst manipulating the ranges, such as `reduceMC()` which not only reduces the Ranges, but collapses the `mcols` into vectors or IRanges::CompressedList objects. Key function from this group are:

- `reduceMC()`, `setdiffMC()`, `intersectMC()`, `unionMC()`, `distinctMC()` and `chopMC()`
- `bestOverlap()` and `propOverlap()` provide simple output easily able to be added as a column within the `mcols` element
- `as_tibble()` coerces a GRanges object to a tibble::tibble.
- `colToRanges()` enables parsing of a single column to a GRanges object, setting all other columns as the `mcols` element.
- `stitchRanges()` merges nearby ranges setting barrier ranges which cannot be crossed when merging
- `partitionRanges()` break apart one set of ranges by another
- `dualFilter()` filters ranges from sliding windows using a guide set of reference ranges where signal is confidently expected
- `mergeByCol()` merges overlapping ranges, as produced by sliding windows
- `mapByFeature()` is able to map a set of GRanges to the most appropriate gene, using any optional combination of promoters, enhancers and HiC interactions
- `grlToSE()` takes selected columns from a GRangesList and sets them as assays within a SummarizedExperiment::RangedSummarizedExperiment object. Used for combining peak intensities or results across multiple ChIP targets.

Visualisation Functions

- `plotHFGC()` is a wrapper to Gviz plotting functions, able to take any combination of HiC, Features, Genes and Coverage (i.e. BigWig) and plot a specified range.
- `plotOverlaps()` visualises overlapping ranges as an UpSet plot or Venn Diagram
- `plotProfileHeatmap()` plots the average signal around a set of ranges, as prepared by `getProfileData()`
- `plotPie()` and `plotSplitDonut()` enable simple comparison across multiple annotation columns within a GRanges object.
- `plotAssayDensities()`, `plotAssayPCA()` and `plotAssayRle()` provide simple interfaces to plotting key values from a SummarizedExperiment::RangedSummarizedExperiment.

Convenience Functions

- `fitAssayDiff()` enables differential signal analysis on a SummarizedExperiment object
- `collapseGenes()` prints a vector of genes for an rmarkdown document, using italics.
- `importPeaks()` imports large numbers of broadPeak or narrowPeak files
- `makeConsensus()` forms consensus peaks from overlapping ranges within a GRangesList()

Author(s)

Stevie Pederson

See Also

Useful links:

- https://github.com/smped/extraChIPs
- Report bugs at https://github.com/smped/extraChIPs/issues

.makeFinalProfileHeatmap

Make a profile heatmap

Description

Make a profile heatmap with optional summary panel at the top

Usage

```
.makeFinalProfileHeatmap(
  data,
  x = NULL,
  y = NULL,
  fill = NULL,
  colour = NULL,
  linetype = NULL,
```

```
  facet_x = NULL,
  facet_y = NULL,
  summary_fun = c("mean", "median", "min", "max", "none"),
  rel_height = 0.3,
  x_lab = NULL,
  y_lab = NULL,
  fill_lab = NULL,
  lab_fun_x = waiver(),
  label_side = c("left", "right", "none"),
  ...
)
```

Arguments

`data`	A data.frame or tibble in long form
`x`, `y`	The values mapped to the x & y axes
`fill`	The column used for heatmap colours
`colour`, `linetype`	Columns used for the summary plot in the top panel
`facet_x`, `facet_y`	Columns used to facet the plot along these axes
`summary_fun`	Function used to create the summary value at each position
`rel_height`	The relative height of the top panel
`x_lab`, `y_lab`, `fill_lab`	_labels added to x/y-axes & the fill legend
`...`	Passed to facet_grid

Details

The workhorse function for generating the final heatmap Expects a single data.frame in long form with requisite columns

Value

A ggplot2 object

`.mapFeatures` *Map ranges to genes using features as an anchor*

Description

Map ranges to genes using features as an anchor

Usage

```
.mapFeatures(.gr, .feat, .genes, .cols, .gr2feat, .feat2gene, ...)
```

Arguments

.gr	The ranges to map onto
.feat	Features to use for mapping
.genes	GRanges object containing gene-level information
.cols	The columns from .genes to map onto .gr
.gr2feat	The maximum distance between ranges and features
.feat2gene	The maximum distance between features & genes
...	Passed to findOverlaps and subsetByOverlaps

Value

A data.frame

.mapGi *Map ranges to genes via Interactions*

Description

Map ranges to genes via Interactions

Usage

```
.mapGi(.gr, .gi, .genes, .cols, .gr2gi, .gi2gene, ...)
```

Arguments

.gr	The ranges to map onto
.gi	GInteractions object
.genes	GRanges object containing gene-level information
.cols	The columns from .genes to map onto .gr
.gr2gi	The maximum distance between ranges and anchors
.gi2gene	The maximum distance between anchors & genes
...	Passed to findOverlaps

Value

data.frame of mapped ranges

.mapWithin *Map ranges to all genes within a set distance*

Description

Map ranges to all genes within a set distance

Usage

```
.mapWithin(.gr, .genes, .cols, .within, ...)
```

Arguments

.gr	The ranges to map onto
.genes	GRanges object containing gene-level information
.cols	The columns from .genes to map onto .gr
.within	The maximum distance between ranges & genes
...	Passed to findOverlaps

Value

A data.frame

addDiffStatus *Add a status column*

Description

Add a status column based on significance and estimated change

Usage

```
addDiffStatus(x, ...)

## S4 method for signature 'data.frame'
addDiffStatus(
  x,
  fc_col = "logFC",
  sig_col = c("FDR", "hmp_fdr", "p_fdr", "adj.P.Value"),
  alpha = 0.05,
  cutoff = 0,
  up = "Increased",
  down = "Decreased",
  other = "Unchanged",
```

```
  missing = "Undetected",
  new_col = "status",
  drop = FALSE,
  ...
)

## S4 method for signature 'DataFrame'
addDiffStatus(x, new_col = "status", ...)

## S4 method for signature 'GRanges'
addDiffStatus(x, ...)

## S4 method for signature 'GRangesList'
addDiffStatus(x, ...)

## S4 method for signature 'SummarizedExperiment'
addDiffStatus(x, ...)
```

Arguments

x	Object to be classified
...	Used to pass arguments between methods
fc_col	Name of the fold-change column
sig_col	Name of the column with significance values
alpha	significance threshold
cutoff	minimum estimated change to be considered in either of the up or down categories
up, down, other	factor levels to annotate regions based on the above criteria
missing	Value to add when either fc_col or sig_col has NA values
new_col	name of the new column to be added
drop	logical(1) Drop unused factor levels from the status column

Details

This takes a simple object and adds a new column classifying entries into one of three categories, as specified using up, down or other. Results in the new column will always be returned as a factor with levels in order of the values provided in the arguments other, down and up

Value

An object of the same type as provided

Examples

```
## Working with a data.frame
set.seed(101)
df <- data.frame(logFC = rnorm(20), p = rbeta(20, shape1 = 1, shape2 = 20))
```

```
df$FDR <- p.adjust(df$p, "fdr")
addDiffStatus(df)

## This works identically with a GRanges object, amongst others
gr <- GRanges(paste0("chr1:", seq_len(20)))
mcols(gr) <- df
addDiffStatus(gr)
```

---

as_tibble *Convert to a tibble*

Description

Convert multiple Genomic objects to tibbles

Usage

```
## S3 method for class 'DataFrame'
as_tibble(x, rangeAsChar = TRUE, ...)

## S3 method for class 'GenomicRanges'
as_tibble(x, rangeAsChar = TRUE, name = "range", ...)

## S3 method for class 'Seqinfo'
as_tibble(x, ...)

## S3 method for class 'GInteractions'
as_tibble(x, rangeAsChar = TRUE, suffix = c(".x", ".y"), ...)

## S3 method for class 'SummarizedExperiment'
as_tibble(x, rangeAsChar = TRUE, ...)

## S3 method for class 'TopTags'
as_tibble(x, ...)
```

Arguments

x	A Genomic Ranges or DataFrame object
rangeAsChar	Convert any GRanges element to a character vector
...	Passed to tibble::as_tibble()
name	Name of column to use for ranges. Ignored if rangeAsChar = FALSE
suffix	Suffix appended to column names for anchor1 and anchor2 of a GInteractions object. Only used if specifying rangeAsChar = FALSE

Details

Quick and dirty conversion into a tibble.

By default, GenomicRanges will be returned with the range as a character column called `range` and all mcols parsed as the remaining columns. Seqinfo information will be lost during coercion.

Given that names for ranges are considered as rownames in the mcols element, these can be simply parsed by setting `rownames = "id"` in the call to `as_tibble()`

When coercing a DataFrame, any Compressed/SimpleList columns will be coerced to S3 list columns. Any GRanges columns will be returned as a character column, losing any additional mcols from these secondary ranges

Coercion of SummarizedExperiment objects will be performed on the `rowRanges()` element, whilst for a GInteractions object, both anchors will returned with the default suffixes `.x` and `.y`

Defined as an S3 method for consistency with existing tidy methods

Value

A tibble

Examples

```
gr <- GRanges("chr1:1-10")
gr$p_value <- runif(1)
names(gr) <- "range1"
gr
as_tibble(gr)
as_tibble(gr, rownames = "id")
as_tibble(mcols(gr))
as_tibble(seqinfo(gr))

hic <- InteractionSet::GInteractions(gr, GRanges("chr1:201-210"))
hic$id <- "interaction1"
as_tibble(hic)
```

`bestOverlap` *Find the best overlap between GRanges*

Description

Find the best overlap between ranges

Usage

```
bestOverlap(x, y, ...)

## S4 method for signature 'GRanges,GRanges'
bestOverlap(
```

```
  x,
  y,
  var = NULL,
  ignore.strand = FALSE,
  missing = NA_character_,
  min_prop = 0.01,
  ...
)

## S4 method for signature 'GRanges,GRangesList'
bestOverlap(
  x,
  y,
  ignore.strand = FALSE,
  missing = NA_character_,
  min_prop = 0.01,
  ...
)
```

Arguments

x	a GRanges object
y	a named GRangesList or GRanges object with mcol as reference category
...	Not used
var	The variable to use as the category. Not required if y is a GRangesList
ignore.strand	logical(1) Passed to findOverlaps
missing	Value to assign to ranges with no overlap
min_prop	Threshold below which overlaps are discarded

Details

This finds the category in the subject GRanges (y) which has the best overlap with the query GRanges (x). The aim is to produce a character vector for best classifying the query GRanges using an external set of features (e.g. promoters, enhancers etc). If the subject (y) is a GRanges object, the values in the specified column will be used as the category. If the subject (y) is a GRangesList, the names of the list will be used to provide the best match

Value

Character vector the same length as the supplied GRanges object

Examples

```
gr <- GRanges("chr1:1-10")
gr_cat <- GRanges(c("chr1:2-10", "chr1:5-10"))
gr_cat$category <- c("a", "b")
propOverlap(gr, gr_cat)
bestOverlap(gr, gr_cat, var = "category")
```

```
grl <- splitAsList(gr_cat, gr_cat$category)
lapply(grl, function(x) propOverlap(gr, x))
bestOverlap(gr, grl)
```

---

centrePeaks *Re-estimate peak centres from coverage*

Description

Use coverage to estimate peak centres

Usage

```
centrePeaks(x, y, ...)

## S4 method for signature 'GRanges,BamFileList'
centrePeaks(
  x,
  y,
  f = c("weighted.cov", "mean", "median"),
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'GRanges,BamFile'
centrePeaks(x, y, ...)

## S4 method for signature 'GRanges,BigWigFileList'
centrePeaks(
  x,
  y,
  f = c("weighted.cov", "mean", "median"),
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'GRanges,BigWigFile'
centrePeaks(x, y, ...)

## S4 method for signature 'GRanges,character'
centrePeaks(x, y, ...)
```

Arguments

x	A set of GRanges representing peaks

y	A suitable set of files with methods defined
...	Used to pass arguments between methods
f	The function to use when estimating a combined peak centre
BPPARAM	An object of class BPPARAM

Details

Use coverage to estimate the centre of a set of peaks or GenomicRanges.

If using the mean or median, the point of maximum coverage for each sample will be found within each peak and these positions will be averaged to return a position representing an estimated peak centre.

If using weighted.cov, positions are weighted by the combined coverage across all samples to return the weighted mean position. In this case coverage will be scaled by total alignments within each bam file before summing across files

Value

A GRanges object with all widths set to one

Examples

```
## Define some peaks
f <- system.file("extdata/peaks.bed.gz", package = "extraChIPs")
peaks <- importPeaks(f, type = "bed")[[1]]
peaks

## Use a bam file to re-centre the regions using highest coverage
bf <- system.file("extdata/bam/ex1.bam", package = "extraChIPs")
centres <- centrePeaks(peaks, bf, BPPARAM = SerialParam())
centres
```

---

chopMC *Keep unique ranges and collapse mcols*

Description

Keep unique ranges by 'chopping' mcols

Usage

```
chopMC(x, simplify = TRUE)
```

Arguments

x	A GenomicRanges object
simplify	logical(1)

Details

This function finds unique ranges and chops **all** mcols in a manner similar to chop. Chopped columns will be returned as `CompressedList` columns, unless `simplify = TRUE` (the default). In this case, columns will be returned as vectors where possible.

Value

A GRanges object

Examples

```
gr <- GRanges(rep(c("chr1:1-10"), 2))
gr$id <- paste0("range", seq_along(gr))
gr$gene <- "gene1"
gr
chopMC(gr)
```

`collapseGenes` *Collapse a vector of gene names*

Description

Collapse a vector of gene names

Usage

```
collapseGenes(
  x,
  sort = TRUE,
  dedup = TRUE,
  format = "_",
  sep = ", ",
  last = " and ",
  numeric = TRUE,
  width = Inf,
  ...
)
```

Arguments

`x`	character vector representing gene names
`sort`	logical(1) Should the names be sorted alphabetically
`dedup`	logical(1) Should duplicate names be removed
`format`	character string for markdown formatting of each element

sep	separator between vector elements
last	character string to place before the last element
numeric	logical(1) sort digits numerically, instead of as strings
width	The maximum width of the string before truncating to ...
...	passed to str_sort

Details

Convenience function to collapse a vector of gene names into a character/glue object of length 1. By default, symbols are deduplicated, sorted alpha-numerically and italicised with an underscore.

Value

a glue object

Examples

```
genes <- c("FOXP3", "BRCA1", "TP53")
collapseGenes(genes)
```

---

colToRanges *Coerce a column to a GRanges object*

Description

Coerce a column to a GRanges object from a rectangular object

Usage

```
colToRanges(x, ...)

## S4 method for signature 'DataFrame'
colToRanges(x, var, seqinfo = NULL, ...)

## S4 method for signature 'GRanges'
colToRanges(x, var, ..., keep_metadata = TRUE)

## S4 method for signature 'data.frame'
colToRanges(x, var, seqinfo = NULL, ...)
```

Arguments

x	A data-frame or GRanges object containing the column to coerce
...	Used to pass arguments to lower-level functions
var	The name of the column to coerce
seqinfo	A seqinfo object to be applied to the new GRanges object. Ignored if the column is already a GRanges object
keep_metadata	logical(1) If the original object is a GRanges object, retain all metadata from the original ranges in the replaced ranges

Details

Take a data.frame-like object and coerce one column to a GRanges object, setting the remainder as the mcols. A particularly useful application of this is when you have a GRanges object with one mcol being a secondary GRanges object.

Alternatively, if you have a data.frame with GRanges represented as a character column, this provides a simple method of coercion. In this case, no Seqinfo element will be applied to the GRanges element.

Value

A GenomicRanges object

Examples

```
set.seed(73)
x <- GRanges(c("chr1:1-10", "chr1:6-15", "chr1:51-60"))
seqinfo(x) <- Seqinfo("chr1", 60, FALSE, "Example")
df <- data.frame(logFC = rnorm(3), logCPM = rnorm(3,8), p = 10^-rexp(3))
mcols(x) <- df
gr <- mergeByCol(x, col = "logCPM", pval = "p")
colToRanges(gr, "keyval_range")
```

---

cytobands *Cytogenetic bands*

Description

Cytogenetic bands for GRCh37/hg19 and GRCh38/hg38

Usage

```
data(grch37.cytobands)

data(grch38.cytobands)
```

Format

Cytogenetic bands for standard chromosomes from GRCh37,in the format required by IdeogramTrack. A data.frame with 5 columns:

chrom Chromosome

chromStart Starting position for each cytogenetic band

chromEnd End position for each cytogenetic band

name Name for each band, e.g. p.36.33

gieStain Staining pattern

An object of class `data.frame` with 862 rows and 5 columns.

Source

https://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/cytoBand.txt.gz

https://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/cytoBand.txt.gz

Examples

```
data(grch37.cytobands)
head(grch37.cytobands)

data(grch38.cytobands)
head(grch38.cytobands)
```

---

defineRegions *Define Genomic Regions Based on Gene Annotations*

Description

Use gene, transcript and exon annotations to define genomic regions

Usage

```
defineRegions(
  genes,
  transcripts,
  exons,
  promoter = c(2500, 500),
  upstream = 5000,
  intron = TRUE,
  proximal = 10000,
  simplify = FALSE,
  cols = c("gene_id", "gene_name", "transcript_id", "transcript_name"),
  ...
)
```

Arguments

genes, transcripts, exons	GRanges objects defining each level of annotation
promoter	Numeric vector defining upstream and/or downstream distances for a promoter. Passing a single value will define a symmetrical promoter The first value represents the upstream range
upstream	The distance from a TSS defining an upstream promoter
intron	logical(1) Separate gene bodies into introns and exons. If intron = FALSE gene bodies will simply be defined as gene bodies
proximal	Distance from a gene to be considered a proximal intergenic region. If set to 0, intergenic regions will simply be considered as uniformly intergenic
simplify	Passed internally to reduceMC and setdiffMC
cols	Column names to be retained from the supplied annotations
...	Not used

Details

Using GRanges annotated as genes, transcripts and exons this function will define ranges uniquely assigned to a region type using a hierarchical process. By default, these region types will be (in order) 1) Promoters, 2) Upstream Promoters, 3) Exons, 4) Introns, 5) Proximal Intergenic and 6) Distal Intergenic.

Setting intron = FALSE will replace introns and exons with a generic "Gene Body" annotation. Setting proximal = 0 will return all intergenic regions (not previously annotated as promoters or upstream promoters) to an "Intergenic" category

Notably, once a region has been defined, it is excluded from all subsequent candidate regions.

Any columns matching the names provided in cols will be returned, and it is assumed that the gene/transcript/exon ranges will contain informative columns in the mcols() element.

Value

A GRangesList

Examples

```
## Define two exons for two transcripts
sq <- Seqinfo(seqnames = "chr1", seqlengths = 50000)
e <- c("chr1:20001-21000", "chr1:29001-29950", "chr1:22001-23000", "chr1:29001-30000")
e <- GRanges(e, seqinfo = sq)
mcols(e) <- DataFrame(
  gene_id = "Gene1", transcript_id = paste0("Trans", c(1, 1, 2, 2))
)

## Define the transcript ranges
t <- unlist(endoapply(split(e, e$transcript_id), range))
t$gene_id <- "Gene1"
t$transcript_id <- names(t)
names(t) <- NULL
```

```
## Summarise to gene level
g <- reduceMC(t)
g$transcript_id <- NA_character_

## Now annotate the regions
regions <- defineRegions(genes = g, transcripts = t, exons = e)
sort(unlist(regions))

## Alternatively, collpse gene body and intergenic ranges
regions <- defineRegions(
  genes = g, transcripts = t, exons = e, intron = FALSE, proximal = 0
)
sort(unlist(regions))
```

`defineSeqinfo` *Use package data to define a Seqinfo object*

Description

Use package data to define a Seqinfo object

Usage

```
defineSeqinfo(
 build = c("GRCh38", "GRCh37", "GRCm39", "GRCm38", "hg19", "hg38", "T2T-CHM13v2.0",
    "mm39", "mm10"),
  chr = TRUE,
  mito,
  ...
)
```

Arguments

`build`	The Genome build used
`chr`	logical(1) Include the prefix "chr"
`mito`	Specify M or MT to include the mitochondrial chromosome. Omitted by default
`...`	Not used

Details

This function will create a Seqinfo object from pre-defined data from the Genome Reference Consortium. Returned objects will always be restricted to assembled molecules only. Currently implemented genome builds represent the four most common builds for ChIP-Seq analysis

Value

A Seqinfo object

Examples

```
defineSeqinfo("GRCh37", TRUE)
defineSeqinfo("GRCh37", FALSE, "MT")
```

`distinctMC` *Keep distinct ranges and mcols*

Description

Keep distinct ranges by including mcols

Usage

```
distinctMC(x, ..., .keep_all = FALSE)
```

Arguments

`x`	A GenomicRanges object
`...`	<data-masking> Passed to distinct
`.keep_all`	If `TRUE`, keep all columns in `x`

Details

Wrapper to distinct for `GRanges` objects. Finds unique ranges and mcols in combination and retains only the distinct combinations, in keeping with the `dplyr` function.

Will default to `unique(granges(x))` if no columns are provided

Value

A GRanges object

Examples

```
gr <- GRanges(rep(c("chr1:1-10"), 2))
gr$id <- paste0("range", seq_along(gr))
gr$gene <- "gene1"
gr
distinctMC(gr)
distinctMC(gr, gene)
distinctMC(gr, gene, .keep_all = TRUE)
```

`dualFilter` *Apply two filters to sliding windows*

Description

Apply two filters to counts generated using sliding windows

Usage

```
dualFilter(
  x,
  bg = NULL,
  ref,
  q = 0.5,
  logCPM = TRUE,
  keep.totals = TRUE,
  bin.size = NULL,
  prior.count = 2,
  BPPARAM = bpparam()
)
```

Arguments

`x`	RangedSummarizedExperiment containing sample counts
`bg`	RangedSummarizedExperiment containing background/input counts, or alternate method for selecting samples from within x, such as a logical, numeric or character vector
`ref`	GRanges object containing ranges where signal is expected
`q`	The upper percentile of the reference ranges expected to be returned when tuning the filtering criteria
`logCPM`	logical(1) Add a logCPM assay to the returned data
`keep.totals`	logical(1) Keep the original library sizes or replace using only the retained windows
`bin.size`	Bin sizes when calling filterWindowsControl. If not specified will default to the largest of 2000bp or 10x the window size
`prior.count`	Passed to filterWindowsControl and filterWindowsProportion
`BPPARAM`	Settings for running in parallel

Details

This function will take sliding (or tiling) windows for it's input as a RangedSummarizedExperiment object. The dual strategy of applying filterWindowsControl and filterWindowsProportion will then be applied. A set of reference ranges for which signal is expected is used to refine the filtering criteria.

Cutoff values are found for both signal relative to input and overall signal, such that the `100*q%` of the (sliding) windows which overlap a reference range will be returned, along with any others which match the dual filtering criteria. In general, higher values of `q` will return more windows as those with weak signal and a marginal overlap with a reference range will be returned. Lower values will ensure that fewer windows, generally with the strongest signal, are retained. Cutoff values for both criteria are added to the metadata element of the returned object.

If setting `bg = NULL` the filterWindowsControl step will be ignored and only the filterWindowsProportion will be used. This should only be performed if no Input sample is available.

Please note that the any `.bam` files referred to in the supplied objects **must** be accessible to this function. It will not run on a separate machine or file structure to that which the original sliding windows were prepared. Please see the example/vignette for runnable code.

Value

A RangedSummarizedExperiment which is a filtered subset of the original object. If requested the assay "logCPM" will be added (`TRUE` by default)

Examples

```
## Taken from the differential_binding vignette
library(tidyverse)
library(Rsamtools)
library(csaw)
library(BiocParallel)
library(rtracklayer)
## For this function we need a set of counts using sliding windows and the
## original BamFiles from which they were taken
## First we'll set up the bam file list
bfl <- system.file(
    "extdata", "bam", c("ex1.bam", "ex2.bam", "input.bam"), package = "extraChIPs"
    ) %>%
    BamFileList() %>%
    setNames(c("ex1", "ex2", "input"))

## Then define the readParam settings for csaw::readParam()
rp <- readParam(
    pe = "none",
    dedup = TRUE,
    restrict = "chr10"
)

## Now we can form our sliding window object with the counts.
wincounts <- windowCounts(
    bam.files = bfl,
    spacing = 60,
    width = 180,
    ext = 200,
    filter = 1,
    param = rp
)
## As this is a subset of reads, add the initial library sizes for accuracy
```

```
## Note that this step is not normally required
wincounts$totals <- c(964076L, 989543L, 1172179L)

## We should also update the metadata for our counts
wincounts$sample <- colnames(wincounts)
wincounts$treat <- as.factor(c("ctrl", "treat", NA))
colData(wincounts)

## The function dualFilter requires a set of peaks which will guide the
## filtering step. This indicate where genuine signal is likely to be found
## and will perform the filtering based on a) signal above the input, and
## b) The overall signal level, using the guide set of peaks to inform the
## cutoff values for inclusion
peaks <- import.bed(
    system.file("extdata", "peaks.bed.gz", package = "extraChIPs")
)
filtcounts <- dualFilter(
    x = wincounts, bg = "input", ref = peaks,
    q = 0.8 # Better to use q = 0.5 on real data
)
filtcounts
```

`ex_datasets` *Datasets for an example region*

Description

Various example datasets for demonstrating analysis and visualisation strategies. Generation of all datasets is documented in `system.file("script/ex_datasets.md", package = "extraChIPs")`

ex_genes Simple GRanges object with complete ranges for each gene

ex_trans Exon & transcript level information prepared for plotting with `Gviz` or `plotHFGC()`

ex_prom Regions defined as promoters

ex_hic Example HiC interactions

Usage

```
data(ex_trans)

data(ex_genes)

data(ex_prom)

data(ex_hic)
```

Format

GRanges and GInteractions objects

All annotations are from GRCh37

An object of class GRanges of length 4.

An object of class GRanges of length 9.

An object of class GInteractions of length 1.

Examples

```
data(ex_trans)
ex_trans
```

`fitAssayDiff` *Detect Differential ChIP Signal*

Description

Detect differential ChIP signal using one of many approaches

Usage

```
fitAssayDiff(x, ...)

## S4 method for signature 'SummarizedExperiment'
fitAssayDiff(
  x,
  assay = "counts",
  design = NULL,
  coef = NULL,
  lib.size = "totals",
  method = c("qlf", "lt", "wald"),
  norm = c("none", "TMM", "RLE", "TMMwsp", "upperquartile"),
  groups = NULL,
  fc = 1,
  lfc = log2(fc),
  asRanges = FALSE,
  offset = NULL,
  weighted = FALSE,
  ...,
  null = c("interval", "worst.case"),
  robust = FALSE,
  type = c("apeglm", "ashr", "normal")
)
```

Arguments

x	a SummarizedExperiment object
...	Passed to normLibSizes and
assay	The assay to use for analysis
design	The design matrix to use for analysis
coef	The required column from the design matrix
lib.size	The column within the colData element which contains the library size information. If set to NULL, column summaries will be used.
method	the analytic method to be used. Can be 'qlf' which will fit counts using the glmQLFit strategy , or 'lt' which fits the limma-trend model on logCPM, or pre-processed logCPM values. Setting method = 'wald' will call nbinomWaldTest
norm	The normalisation strategy to use when running the glmQLF model or the Wald test. The value 'none' relies solely on library-size normalisation, and is the default. All methods available in normLibSizes are implemented. Ignored when using method = "lt"
groups	character(1) If a column name is supplied here, group-based normalisation will be applied to GLM models treating data in this column as a grouping factor. Ignored when using method = "lt"
fc, lfc	Thresholds passed to treat, glmTreat or lfcShrink
asRanges	logical(1). By default, the returned object will be a SummarizedExperiment object with the results added to the rowData element. Setting asRanges = TRUE will only return the GRanges object from this element
offset	If provided will be used as the offset when a DGEList object is created during model fitting for method = 'qlf'
weighted	logical(1) Passed to normLibSizes. Only used when applying a TMM-type normalisation strategy
null	Passed to glmTreat glmQLFit when method = "qlf". If method = "lt", instead passed to lmFit
robust	Passed to treat and eBayes
type	Passed to lfcShrink

Details

Starting with a SummarizedExperiment object this function fits either a glmQLFit model to count data, performs the nbinomWaldTest on count data, or applies the limma-trend model to logCPM data.

If fitting Generalised Linear Models via glmQLFit, options for normalisation are "none", which normalises to library size. Existing library sizes are commonly found in the "totals" column of the colData element and this is attempted by default. All methods provided in normLibSizes are also implemented, with the added possibility of normalising within groups instead of across the entire dataset. To enable this, the column with the grouping factor is expected to be in the colData element and is simply called by column name. No normalisation is applied when using the limma-trend model, as this allows for previous normalisation strategies to be performed on the data.

If testing with nbinomWaldTest, applying RLE normalisation without groups, and using colSums for library sizes (instead of total alignments), the standard normalisation factors from `DESeq2::estimateSizeFactors` will be used. In all other scenarios, normalisation factors as returned by normLibSizes will be used. The fitType is set to 'local' when estimating dispersions, and this can be easily modified by passing fitType via the dot arguments. Results are additionally returned after applying lfcShrink, including the svalue returned by this approach.

Normalising to ChIP Input samples is not yet implemented. Similarly, the use of offsets when applying the Wald test is not yet implemented.

Range-based hypothesis testing is implemented using glmTreat or treat. Setting fc to 1 (or lfc to 0) will default to a point-based null hypothesis, equivalent to either glmQLFTest (method = "qlf") or eBayes (method = "lt"). When applying nbinomWaldTest, lfcShrink will be applied.

It should also be noted that this is primarily a convenience function and if requiring intermediate output from any steps, then these can be run individually as conventionally specified.

Value

A SummarizedExperiment object with results set as the `rowData` element. Any existing columns not contained in the differential ChIP results will be retained. Results from testing will contain logCPM, logFC, PValue and any t/F statistic as appropriate, along with an FDR-adjusted p-value.

If specifying a range-based H0 by setting lfc != 0, an additional column p_mu0 will be included which is the p-value for the point H0: logFC = 0. These are not used for FDR-adjusted p-values but can be helpful when integrating multiple ChIP targets due to the increase in false-negatives when using a range-based H0, and when requiring more accurate identification of truly unchanged sites, as opposed to those which simply fail to achieve significance using a range-based H0 where arbitrary cutoff values are used.

Examples

```
nrows <- 200; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
colnames(counts) <- paste0("Sample_", seq_len(ncols))
df <- DataFrame(treat = c("A", "A", "A", "B", "B", "B"))
df$treat <- as.factor(df$treat)
se <- SummarizedExperiment(
  assays = SimpleList(counts = counts), colData = df
)
X <- model.matrix(~treat, colData(se))
se <- fitAssayDiff(se, design = X, lib.size = NULL)
rowData(se)
```

`fixed_width_datasets` *Datasets for the Fixed-Width Vignette*

Description

GRangesList of peaks and SummarizedExperiment of counts All were saved during initial vignette preparation at https://github.com/smped/extraChIPs_vignette/blob/main/differential_signal_fixed.Rmd

Usage

```
data(se)

data(peaks)
```

Format

An object of class `RangedSummarizedExperiment` with 188 rows and 6 columns.

An object of class `CompressedGRangesList` of length 6.

Examples

```
data(se)
se
data(peaks)
peaks
```

`getProfileData` *Get Profile Data surrounding specified ranges*

Description

Get coverage Profile Data surrounding specified ranges

Usage

```
getProfileData(x, gr, ...)

## S4 method for signature 'BigWigFile,GenomicRanges'
getProfileData(
  x,
  gr,
  upstream = 2500,
  downstream = upstream,
  bins = 100,
  mean_mode = "w0",
  log = TRUE,
  offset = 1,
  n_max = Inf,
  ...
)
```

```
## S4 method for signature 'BigWigFileList,GenomicRanges'
getProfileData(
  x,
  gr,
  upstream = 2500,
  downstream = upstream,
  bins = 100,
  mean_mode = "w0",
  log = TRUE,
  offset = 1,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'character,GenomicRanges'
getProfileData(
  x,
  gr,
  upstream = 2500,
  downstream = upstream,
  bins = 100,
  mean_mode = "w0",
  log = TRUE,
  offset = 1,
  ...
)
```

Arguments

x	A BigWigFile or BigWiFileList
gr	A GRanges object
...	Passed to normalizeToMatrix
upstream	The distance to extend upstream from the centre of each range within gr
downstream	The distance to extend downstream from the centre of each range within gr
bins	The total number of bins to break the extended ranges into
mean_mode	The method used for calculating the score for each bin. See normalizeToMatrix for details
log	logical(1) Should the returned values be log2-transformed
offset	Value added to data if log-transforming. Ignored otherwise
n_max	Upper limit on the number of ranges to return profile data for. By default, no limit will be applied .
BPPARAM	Passed internally to bplapply

Details

This will take all provided ranges and set as identical width ranges, extending by the specified amount both up and downstream of the centre of the provided ranges. By default, the ranges exten-

sions are symmetrical and only the upstream range needs to be specified, however this parameterisation allows for non-symmetrical ranges to be generated.

These uniform width ranges will then be used to extract the value contained in the score field from one or more BigWigFiles. Uniform width ranges are then broken into bins of equal width and the average score found within each bin.

The binned profiles are returned as a DataFrameList called `profile_data` as a column within the resized GRanges object. Column names in each DataFrame are `score`, `position` and bp.

If passing a BigWigFileList, profiles will be obtained in series by default. To run in parallel pass a MulticoreParam object to the `BPPARAM` argument.

Value

GRanges or GrangesList with column profile_data, as described above

Examples

```
bw <- system.file("tests", "test.bw", package = "rtracklayer")
gr <- GRanges("chr2:1000")
pd <- getProfileData(bw, gr, upstream = 500, bins = 10)
pd
pd$profile_data
```

---

grlToSE *Set columns from a GRangesList as Assays in a SummarizedExperiment*

Description

Move one or more columns from a GRangesList elements into assays in a RangesSummarizedExperiment

Usage

```
grlToSE(x, ...)

## S4 method for signature 'GRangesList'
grlToSE(
  x,
  assayCols = c(),
  metaCols = c(),
  keyvals = c(),
  by = c("min", "max"),
  ...,
  ignore.strand = FALSE
)
```

Arguments

x	A GrangesList
...	Passed to reduce
assayCols	Columns to move to separate assays
metaCols	Columns to move to mcols within the rowRanges element
keyvals	The value to use when choosing representative values
by	How to choose by keyvals
ignore.strand	logical(1). Whether the strand of the input ranges should be ignored or not.

Details

Given a GRangesList which would commonly represent multiple samples, reduce any overlapping ranges into a consensus range, setting any metadata columns to be retained as separate assays. These columns may contain values such as coverage, p-values etc.

Additional columns can also be placed as rowData columns where the original values are better suited to information about the consensus range rather than the sample (or GRangesList element).

Only one value for each range will be retained, and these are chosen using the value provided as the keyvals, taking either the min or max value in this column as the representative range.

Value

A RangedSummarizedExperiment

Examples

```
a <- GRanges("chr1:1-10")
a$feature <- "Gene"
a$p <- 0.1
b <- GRanges(c("chr1:6-15", "chr1:15"))
b$feature <- c("Gene", "Promoter")
b$p <- c(0.5, 0.01)
grl <- GRangesList(a = a, b = b)
grl
se <- grlToSE(
  grl, assayCols = "p", metaCols = "feature", keyvals = "p", by = "min"
)
assay(se, "p")
rowRanges(se)
```

`importPeaks` *Import peaks*

Description

Import peaks in narrowPeak, broadPeak or bed format

Usage

```
importPeaks(
  x,
  type = c("narrow", "broad", "bed"),
  blacklist,
  seqinfo,
  pruning.mode = c("coarse", "error"),
  sort = TRUE,
  setNames = TRUE,
  glueNames = "{basename(x)}",
  centre = FALSE,
  nameRanges = TRUE,
  ...
)
```

Arguments

`x`	One or more files to be imported. All files must be of the same type, i.e. narrow or broad
`type`	The type of peaks to be imported
`blacklist`	A set of ranges to be excluded
`seqinfo`	A seqinfo object to be applied to the GRanges objects
`pruning.mode`	How to handle conflicts if supplying a seqinfo object. Defaults to `pruning.mode = "coarse"`. Only "coarse" and "error" are implemented. See seqinfo.
`sort`	logical. Should the ranges be sorted during import
`setNames`	logical Set basename(x) as the name for each element of the GRangesList
`glueNames`	glue syntax for naming list elements
`centre`	Add the estimated peak centre. Ignored unless type = "narrow"
`nameRanges`	Place any values in the name column as range names within each file.
`...`	passed to `sort`

Details

Peaks are imported from narrowPeak, broadPeak or bed format as GenomicRanges objects.

If importing bed files, only the default 3-6 columns will imported.

Value

A GRangesList

Examples

```
fl <- system.file(
    c("extdata/ER_1.narrowPeak", "extdata/ER_2.narrowPeak"),
    package = "extraChIPs"
)
peaks <- importPeaks(fl)
peaks
```

`makeConsensus` *Make a set of consensus peaks*

Description

Make a set of consensus peaks based on the number of replicates

Usage

```
makeConsensus(
  x,
  p = 0,
  var = NULL,
  method = c("union", "coverage"),
  ignore.strand = TRUE,
  simplify = FALSE,
  min_width = 0,
  merge_within = 1L,
  ...
)
```

Arguments

`x`	A GRangesList
`p`	The minimum proportion of samples (i.e. elements of `x`) required for a peak to be retained in the output. By default all merged peaks will be returned
`var`	Additional columns in the `mcols` element to retain
`method`	Either return the union of all overlapping ranges, or the regions within the overlapping ranges which are covered by the specified proportion of replicates. When using p = 0, both methods will return identical results
`ignore.strand, simplify, ...`	Passed to reduceMC or intersectMC internally
`min_width`	Discard any regions below this width
`merge_within`	Passed to reduce as `min.gapwidth`

Details

This takes a list of GRanges objects and forms a set of consensus peaks.

When using method = "union" the union ranges of all overlapping peaks will be returned, using the minimum proportion of replicates specified. When using method = "coverage", only the regions within each overlapping range which are 'covered' by the minimum proportion of replicates specified are returned. This will return narrower peaks in general, although some artefactual very small ranges may be included (e.g. 10bp). Careful setting of the min_width and merge_within parameters may be very helpful for these instances. It is also expected that setting method = "coverage" should return the region within each range which is more likely to contain the true binding site for the relevant ChIP targets

Value

`GRanges` object with mcols containing a logical vector for every element of x, along with the column `n` which adds all logical columns. These columns denote which replicates contain an overlapping peak for each range

If any additional columns have been requested using `var`, these will be returned as CompressedList objects as produced by `reduceMC()` or `intersectMC()`.

See Also

reduceMC intersectMC

Examples

```
data("peaks")
## The first three replicates are from the same treatment group
grl <- peaks[1:3]
names(grl) <- gsub("_peaks.+", "", names(grl))
makeConsensus(grl)
makeConsensus(grl, p = 2/3, var = "score")

## Using method = 'coverage' finds ranges based on the intersection
makeConsensus(grl, p = 2/3, var = "score", method = "coverage")
```

`mapByFeature` *Map Genomic Ranges to genes using defined features*

Description

Map Genomic Ranges to genes using defined regulatory features

Usage

```
mapByFeature(
  gr,
  genes,
  prom,
  enh,
  gi,
  cols = c("gene_id", "gene_name", "symbol"),
  gr2prom = 0,
  gr2enh = 0,
  gr2gi = 0,
  gr2gene = 1e+05,
  prom2gene = 0,
  enh2gene = 1e+05,
  gi2gene = 0,
  ...
)
```

Arguments

gr	GRanges object with query ranges to be mapped to genes
genes	GRanges object containing genes (or any other nominal feature) to be assigned
prom	GRanges object defining promoters
enh	GRanges object defining Enhancers
gi	GInteractions object defining interactions. Mappings from interactions to genes should be performed as a separate prior step.
cols	Column names to be assigned as mcols in the output. Columns must be minimally present in genes. If all requested columns are found in any of prom, enh or gi, these pre-existing mappings will be preferentially used. Any columns not found in utilised reference objects will be ignored.
gr2prom	The maximum permissible distance between a query range and any ranges defined as promoters
gr2enh	The maximum permissible distance between a query range and any ranges defined as enhancers
gr2gi	The maximum permissible distance between a query range and any ranges defined as GInteraction anchors
gr2gene	The maximum permissible distance between a query range and genes (for ranges not otherwise mapped)
prom2gene	The maximum permissible distance between a range provided in prom and a gene
enh2gene	The maximum permissible distance between a range provided in enh and a gene
gi2gene	The maximum permissible distance between a GInteractions anchor (provided in gi) and a gene
...	Passed to findOverlaps and overlapsAny internally

Details

This function is able to utilise feature-level information and long-range interactions to enable better mapping of regions to genes. If provided, this essentially maps from ranges to genes using the regulatory features as a framework. The following sequential strategy is used:

1. Ranges overlapping a promoter are assigned to that gene
2. Ranges overlapping an enhancer are assigned to **all genes** within a specified distance
3. Ranges overlapping a long-range interaction are assigned to all genes connected by the interaction
4. Ranges with no gene assignment from the previous steps are assigned to *all overlapping genes* or the nearest gene within a specified distance

If information is missing for one of these steps, the algorithm will simply proceed to the next step. If no promoter, enhancer or interaction data is provided, all ranges will be simply mapped by step 4. Ranges can be mapped by any or all of the first three steps, but step 4 is mutually exclusive with the first 3 steps.

Distances between each set of features and the query range can be individually specified by modifying the `gr2prom`, `gr2enh`, `gr2gi` or `gr2gene` parameters. Distances between features and genes can also be set using the parameters `prom2gene`, `enh2gene` and `gi2gene`.

Additionally, if previously defined mappings are included with any of the `prom`, `enh` or `gi` objects, this will be used in preference to any obtained from the `genes` object.

Value

A GRanges object with added mcols as specified

Examples

```
## Define some genes
genes <- GRanges(c("chr1:2-10:*", "chr1:25-30:-", "chr1:31-40:+"))
genes$gene_id <- paste0("gene", seq_along(genes))
genes
## Add a promoter for each gene
prom <- promoters(genes, upstream = 1, downstream = 1)
prom
## Some ranges to map
gr <- GRanges(paste0("chr1:", seq(0, 60, by = 15)))
gr

## Map so that any gene within 25bp of the range is assigned
mapByFeature(gr, genes, gr2gene = 25)

## Now use promoters to be more accurate in the gene assignment
## Given that the first range overlaps the promoter of gene1, this is a
## more targetted approach. Similarly for the third range
mapByFeature(gr, genes, prom, gr2gene = 25)
```

`mapGrlCols` *Collapse a GRangesList adding multiple columns from each element*

Description

Make consensus peaks and add individual columns from each original GRangesList element

Usage

```
mapGrlCols(
  x,
  var = NULL,
  collapse = NULL,
  collapse_sep = "; ",
  name_sep = "_",
  include = FALSE,
  ...
)
```

Arguments

`x`	GRangesList
`var`	Column(s) to map onto the set of consensus peaks
`collapse`	Columns specified here will be simplified into a single column. Should only be character or factor columns
`collapse_sep`	String to separate values when collapsing columns
`name_sep`	String to separate values when adding column names
`include`	logical(1) Include the original ranges as character columns
`...`	Passed to makeConsensus

Details

Starting with a GRangesList, make a single GRanges object with select columns from each element added to the new object

Value

GRanges object with a set of consensus ranges across all list elements and values from each element mapped to these consensus ranges.

If requested (`include = TRUE`) the original ranges are returned as character columns, as there will be multiple NA values in each.

Examples

```
a <- GRanges(paste0("chr1:", seq(1, 61, by = 20)))
width(a) <- 5
a$logFC <- rnorm(length(a))
a_g <- as.list(paste("Gene", seq_along(a)))
a_g[[1]] <- c("Gene 0", a_g[[1]])
a$genes <- as(a_g, "CompressedList")

b <- GRanges("chr1:61-70")
b$logFC <- rnorm(1)
b$genes <- as(list("Gene 5"), "CompressedList")

grl <- GRangesList(A = a, B = b)
mapGrlCols(grl, var = "logFC")

## This forms a union of overlapping rangesby default
## Pass methods to makeConsensus() to change to regions with coverage == 2
mapGrlCols(grl, var = "logFC", method = "coverage", p = 1)

## Columns can be collapsed to merge into a single column
mapGrlCols(grl, var = "logFC", collapse = "genes")

## Original ranges can also be included
mapGrlCols(grl, collapse = "genes", include = TRUE)
```

`mergeByCol` *Merge sliding windows using a specified column*

Description

Merge sliding windows using a specified column

Usage

```
mergeByCol(x, ...)

## S4 method for signature 'GenomicRanges'
mergeByCol(
  x,
  df = NULL,
  col,
  by = c("max", "median", "mean", "min"),
  logfc = "logFC",
  pval = "P",
  inc_cols,
  p_adj_method = "fdr",
```

```
  merge_within = 1L,
  ignore_strand = TRUE,
  min_win = 1,
  ...
)

## S4 method for signature 'RangedSummarizedExperiment'
mergeByCol(
  x,
  df = NULL,
  col,
  by = c("max", "median", "mean", "min"),
  logfc = "logFC",
  pval = "P",
  inc_cols,
  p_adj_method = "fdr",
  merge_within = 1L,
  ignore_strand = FALSE,
  ...
)
```

Arguments

x	A GenomicRanges or SummarizedExperiment object
...	Not used
df	A data.frame-like object containing the columns of interest. If not provided, any columns in the mcols() slot will be used.
col	The column to select as representative of the merged ranges
by	The method for selecting representative values
logfc	Column containing logFC values
pval	Column containing p-values
inc_cols	Any additional columns to return. Output will always include columns specified in the arguments `col`, `logfc` and `pval`. Note that values from any additional columns will correspond to the selected range returned in keyval_range
p_adj_method	Any of p.adjust.methods
merge_within	Merge any ranges within this distance
ignore_strand	Passed internally to reduce and findOverlaps
min_win	Only keep merged windows derived from at least this number

Details

This merges sliding windows using the values in a given column to select representative values for the subsequent merged windows. Values can be chosen from the specified column using any of `min()`, `max()`, `mean()` or `median()`, although `max()` is strongly recommended when specifying values like logCPM. Once a representative range is selected using the specified column, values from columns specified using `inc_cols` are also returned. In addition to these columns, the range

from the representative window is returned in the mcols element as a GRanges object in the column `keyval_range`.

Merging windows using either the logFC or p-value columns is not implemented.

If adjusted p-values are requested an additional column names the same as the initial p-value, but tagged with the adjustment method, will be added. In addition, using the p-value from the selected window, the number of windows with lower p-values are counted by direction and returned in the final object. The selected window will always be counted as up/down regardless of significance as the p-value for this column is taken as the threshold. This is a not dissimilar approach to cluster-direction.

If called on a SummarizedExperiment object, the function will be applied to the `rowRanges` element.

Value

A Genomic Ranges object

Examples

```
x <- GRanges(c("chr1:1-10", "chr1:6-15", "chr1:51-60"))
set.seed(1001)
df <- DataFrame(logFC = rnorm(3), logCPM = rnorm(3,8), p = rexp(3, 10))
mergeByCol(x, df, col = "logCPM", pval = "p")
mcols(x) <- df
x
mergeByCol(x, col = "logCPM", pval = "p")
```

---

mergeByHMP *Merge Sliding Windows using the Harmonic Mean P*

Description

Merge overlapping windows using harmonic mean p-values from significance testing

Usage

```
mergeByHMP(x, ...)

## S4 method for signature 'GenomicRanges'
mergeByHMP(
  x,
  df = NULL,
  w = NULL,
  logfc = "logFC",
  pval = "P",
  cpm = "logCPM",
  inc_cols = NULL,
  p_adj_method = "fdr",
```

```
  merge_within = 1L,
  ignore_strand = TRUE,
  min_win = 1,
  keyval = c("min", "merged"),
  hm_pre = "hm",
  ...
)

## S4 method for signature 'RangedSummarizedExperiment'
mergeByHMP(
  x,
  df = NULL,
  w = NULL,
  logfc = "logFC",
  pval = "P",
  cpm = "logCPM",
  inc_cols = NULL,
  p_adj_method = "fdr",
  merge_within = 1L,
  ignore_strand = FALSE,
  hm_pre = "hm",
  ...
)
```

Arguments

x	GenomicRanges object
...	Not used
df	data.frame with results of differential binding analysis performed using a sliding window strategy. If not provided, the columns in the mcols() element of x will be used
w	vector of weights to applied when calculating harmonic mean p-values
logfc, pval, cpm	Column names for the values holding window specific estimates of change in binding (logfc), overall signal intensity (cpm) and the significance from statistical testing (pval).
inc_cols	(Optional) Character vector of any additional columns in df to return. Values will correspond to the range in the keyval_range column
p_adj_method	One of p.adjust.methods or "fwer". If "fwer" is specified the adjusted harmonic-mean p-value will be returned in a form which strictly controls the experiment-wide FWER. Please see vignette("harmonicmeanp") for more details
merge_within	Merge any non-overlapping windows within this distance
ignore_strand	Passed internally to reduce and findOverlaps
min_win	Only keep merged windows derived from at least this number
keyval	Return the key-value range as the window associated with the minimum p-value, or by merging the ranges from all windows with raw p-values below the merged harmonic-mean p-value
hm_pre	Prefix to add to the beginning of all HMP-derived columns

Details

When using sliding windows to test for differential signal, overlapping windows can be merged based on the significance of results. `mergeByHMP()` merges overlapping windows using the asymptotically exact harmonic mean p-value p.hmp from the individual, window-level tests. This tests the Null Hypothesis that there is no significance amongst the initial set of p-values, and returns a summarised value which controls the FDR within a set of tests (Wilson, PNAS, 2019). Multilevel testing across the set of results is currently implemented using `p_adj_method = "fwer"`

Given that the harmonic mean p-value is calculated from the inverse p-values, these are used to provide a *weighted average* of expression and logFC values in the returned object. Any weights provided in `w` are ignored for these values as they are simple representative estimates. The representative range returned in `keyval_range` corresponds to the window with the lowest p-value.

The total number of windows is also returned in the final object, with the summarised values n_up and n_down indicating the number of windows with raw p-values below the calculated harmonic mean p-value, and with the corresponding direction of change.

The column containing the harmonic mean p-values is returned as 'hmp'. An additional column with adjusted hmp-values is returned with the suffix '_*' added where the p-value adjustment method is added after the underscore.

Value

A GenomicRanges object with merged ranges from the original object along with summarised or representative values from the relevant columns. The range corresponding to a representative values is also returned as described above

Examples

```
x <- GRanges(c("chr1:1-10", "chr1:6-15", "chr1:51-60"))
set.seed(1001)
df <- DataFrame(logFC = rnorm(3), logCPM = rnorm(3,8), p = rexp(3, 10))
mergeByHMP(x, df, pval = "p")
mcols(x) <- df
x
mergeByHMP(x, pval = "p", p_adj_method = "fwer")
```

`mergeBySig` *Merge overlapping ranges based on p-values*

Description

Merge overlapping windows using p-values from significance testing

Usage

```
mergeBySig(x, ...)

## S4 method for signature 'GenomicRanges'
mergeBySig(
  x,
  df = NULL,
  logfc = "logFC",
  pval = "P",
  cpm = "logCPM",
  inc_cols,
  p_adj_method = "fdr",
  alpha = 0.05,
  method = c("combine", "best", "minimal"),
  merge_within = 1L,
  ignore_strand = TRUE,
  min_win = 1,
  ...
)

## S4 method for signature 'RangedSummarizedExperiment'
mergeBySig(
  x,
  df = NULL,
  logfc = "logFC",
  pval = "P",
  cpm = "logCPM",
  inc_cols,
  p_adj_method = "fdr",
  alpha = 0.05,
  method = c("combine", "best", "minimal"),
  merge_within = 1L,
  ignore_strand = TRUE,
  ...
)
```

Arguments

x	GenomicRanges object
...	Passed to all csaw functions being wrapped
df	data.frame with results of differential binding analysis performed using a sliding window strategy. If not provided, the columns in the mcols() element of x will be used
logfc, pval, cpm	Column names for the values holding window specific estimates of change in binding (logfc), overall signal intensity (cpm) and the significance from statistical testing (pval)
inc_cols	(Optional) Character vector of any additional columns in df to return

p_adj_method	One of p.adjust.methods
alpha	Significance threshold to apply during internal calculations
method	Shorthand versions for which csaw strategy to use for merging windows. Choose from 'combine' (combineTests), 'best' (getBestTest) or 'minimal' (minimalTests).
merge_within	Merge any non-overlapping windows within this distance
ignore_strand	Passed internally to reduce and findOverlaps
min_win	Only keep merged windows derived from at least this number

Details

When using sliding windows to test for differential signal, overlapping windows can be merged based on the significance of results. mergeBySig() is a wrapper to the functions combineTests, getBestTest and minimalTests, using each function's approach to finding a representative window. The returned object differs from those returned by the original functions in that the description of windows as 'up', 'down' or mixed is omitted and the genomic range corresponding to the representative window is also returned. Column names also correspond to those in the original object.

An additional column with adjusted p-values is returned. This column retains the same name as the original but with the suffix '_*' added where the p-value adjustment method is added after the underscore.

Value

A GenomicRanges object with overlapping ranges from the original object merged and representative values returned. The range corresponding to the representative values is also returned

Examples

```
x <- GRanges(c("chr1:1-10", "chr1:6-15", "chr1:51-60"))
set.seed(1001)
df <- DataFrame(logFC = rnorm(3), logCPM = rnorm(3,8), p = rexp(3, 10))
mcols(x) <- df
mergeBySig(x, pval = "p", method = "combine")
mergeBySig(x, pval = "p", method = "best")
mergeBySig(x, pval = "p", method = "min")
```

---

partitionRanges *Partition a set of Genomic Ranges*

Description

Partition a set of Genomic Ranges by another

Usage

```
partitionRanges(x, y, ...)

## S4 method for signature 'GRanges,GRanges'
partitionRanges(
  x,
  y,
  y_as_both = TRUE,
  ignore.strand = FALSE,
  simplify = TRUE,
  suffix = c(".x", ".y"),
  ...
)
```

Arguments

x, y	GenomicRanges objects
...	Not used
y_as_both	logical(1) If there are any unstranded regions in y, should these be assigned to both strands. If TRUE unstranded regions can be used to partition stranded regions
ignore.strand	If set to TRUE, then the strand of x and y is set to "*" prior to any computation.
simplify	Pass to chopMC and simplify mcols in the output
suffix	Added to any shared column names in the provided objects

Details

The query set of ranges can be broken in regions which strictly overlap a second set of ranges. The complete set of mcols from both initial objects will included in the set of partitioned ranges

Value

A GRanges object

Examples

```
x <- GRanges(c("chr1:1-10", "chr1:6-15"))
x$id <- paste0("range", seq_along(x))
x
y <- GRanges(c("chr1:2-5", "chr1:6-12"))
y$id <- paste0("range", seq_along(y))
y
partitionRanges(x, y)
```

`plotAssayDensities` *Plot Densities for any assay within a SummarizedExperiment*

Description

Plot Densities for any assay within a SummarizedExperiment

Usage

```
plotAssayDensities(x, ...)

## S4 method for signature 'SummarizedExperiment'
plotAssayDensities(
  x,
  assay = "counts",
  group = NULL,
  colour = NULL,
  fill = NULL,
  linetype = NULL,
  linewidth = NULL,
  alpha = NULL,
  trans = NULL,
  n_max = Inf,
  ...
)
```

Arguments

`x`	A SummarizedExperiment object
`...`	Passed to geom_density
`assay`	An assay within x
`group`	Used by geom_line. Defaults to the column names but treatment groups can also be specified to summarise within groups
`colour`, `fill`, `alpha`	Optional column in colData to set the respective aesthetics. Can also be any valid colour specification as a fixed value or a fixed alpha value
`linetype`, `linewidth`	Any optional column in colData used to determine linetype or linewidth. Can also be fixed values
`trans`	character(1). Any transformative function to be applied to the data before calculating the density, e.g. `trans = "log2"`
`n_max`	Maximum number of points to use when calculating densities

Details

Uses ggplot2 to create a density plot for all samples within the selected assay

Value

A ggplot2 object. Scales and labels can be added using conventional ggplot2 syntax.

Examples

```
data("se")
se$treatment <- c("E2", "E2", "E2", "E2DHT", "E2DHT", "E2DHT")
## Plot individual samples
plotAssayDensities(se, colour = "treatment")
## Plot combined within treatment groups
plotAssayDensities(se, colour = "treatment", group = "treatment")
## Use a data transformation
plotAssayDensities(se, trans = "log1p", colour = "treat")
```

---

plotAssayHeatmap *Draw a heatmap from a single SummarizedExperiment assay*

Description

Use ggplot2 to create a heatmap from a SummarizedExperiment object

Usage

```
plotAssayHeatmap(x, ...)

## S4 method for signature 'SummarizedExperiment'
plotAssayHeatmap(
  x,
  assay = "counts",
  by_x = "colnames",
  facet_x = NULL,
  ysideline = FALSE,
  yside_col = NULL,
  trans = NULL,
  n_max = 100,
  ...
)
```

Arguments

x	a SummarizedExperiment object
...	Not used
assay	the assay to take values from
by_x	the parameter to use for the x-axis. Will default to column names but should be one value per sample, such as an additional column containing shortened sample labels.

facet_x	column from colData(x) which will be used to group samples along the x-axis
ysideline	logical(1) Draw a line across the side of the y-axis summarising values for each range
yside_col	column from colData(x) to group and colour the lines drawn on the side of the y-axis. If grouping by treatment or replicate, the mean values will be shown
trans	character(1). Any transformative function to be applied to the data before calculating the density, e.g. `trans = "log2"`
n_max	Maximum number of ranges to draw

Details

Draw a heatmap containing selected values from an assay within a SummarizedExperiment object. Columns within the colData element of the object can be used to facet along the x-axis (e.g. treatment groups). The maximum number of points is set to be 100, although this can be changed easily should the plot require more ranges to be drawn.

The averages across any grouping of samples can be drawn as a line plot on the side of the y-axis by setting `ysideline = TRUE`, with groups as specified in `yside_col`. This feature is added for the specific context of neighbouring or overlapping ranges, and as such may be less informative in any other scenario

The returned object is a ggplot2 object so scales can easily be added after heatmap creation using scale_fill_* for the main heatmap, and scale_colour_* for any groupings along the y-axis

Value

A `ggplot2` object. Scales and labels can be added using conventional `ggplot2` syntax.

Examples

```
nrows <- 10; ncols <- 4
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
colnames(counts) <- paste0("Sample_", seq_len(ncols))
df <- DataFrame(treat = c("A", "A", "B", "B"))
se <- SummarizedExperiment(
  assays = SimpleList(counts = counts),
  colData = df
)
rowRanges(se) <- GRanges(paste0("chr1:", seq_len(nrows)))
plotAssayHeatmap(se, facet_x = "treat")
```

`plotAssayPCA` *Plot PCA For any assay within a SummarizedExperiment*

Description

Plot PCA for any assay within a SummarizedExperiment object

Usage

```
plotAssayPCA(x, ...)

## S4 method for signature 'SummarizedExperiment'
plotAssayPCA(
  x,
  assay = "counts",
  colour = NULL,
  shape = NULL,
  size = NULL,
  fill = NULL,
  label = "colnames",
  show_points = TRUE,
  pc_x = 1,
  pc_y = 2,
  trans = NULL,
  n_max = Inf,
  tol = sqrt(.Machine$double.eps),
  rank = NULL,
  ...
)
```

Arguments

`x`	An object containing an assay slot
`...`	Passed to geom_text and geom_point
`assay`	The assay to perform PCA on
`colour, size`	The column names to be used for colours and point/label size. Can be fixed values (e.g. size = 3) and can also be a manipulation of a column, e.g. colour = log10(totals)
`shape, fill`	The column name(s) to be used for determining the shape or fill colour of plotted points. Can also be a fixed value
`label`	The column name to be used for labels. Will default to the column names of the SummarizedExperiment
`show_points`	logical(1). Display the points. If `TRUE` any labels will repel. If `FALSE`, labels will appear at the exact points
`pc_x`	numeric(1) The PC to plot on the x-axis

pc_y	numeric(1) The PC to plot on the y-axis
trans	character(1). Any transformative function to be applied to the data before performing the PCA, e.g. trans = "log2"
n_max	Subsample the data to this many points before performing PCA
tol, rank	Passed to prcomp

Details

Uses ggplot2 to create a PCA plot for the selected assay. Any numerical transformation prior to performing the PCA can be specified using the trans argument

Value

A ggplot2 object

Examples

```
data("se")
se$treatment <- c("E2", "E2", "E2", "E2DHT", "E2DHT", "E2DHT")
se$sample <- colnames(se)
plotAssayPCA(se, trans = "log1p", colour = "treatment", label = "sample")
plotAssayPCA(
  se, trans = "log1p", colour = "treatment", label = "sample",
  size = log10(totals), shape = 17
)
plotAssayPCA(
  se, trans = "log1p", colour = "treatment", label = "sample",
  show_points = FALSE
)
```

---

plotAssayRle *Plot RLE for a given assay within a SummarizedExperiment*

Description

Plot RLE for a given assay within a SummarizedExperiment

Usage

```
plotAssayRle(x, ...)

## S4 method for signature 'SummarizedExperiment'
plotAssayRle(
  x,
  assay = "counts",
  colour = NULL,
```

```
  fill = NULL,
  rle_group = NULL,
  by_x = "colnames",
  n_max = Inf,
  trans = NULL,
  ...
)
```

Arguments

x	A SummarizedExperiment object
...	Passed to geom_boxplot
assay	The assay to plot
colour	Column from colData(x) to outline the boxplots
fill	Column from colData(x) to fill the boxplots
rle_group	Column from colData(x) to calculate RLE within groups Commonly an alternative sample label.
by_x	Boxplots will be drawn by this grouping variable from colData(x). If not specified, the default values will be colnames(x)
n_max	Maximum number of points to plot
trans	character(1). Numerical transformation to apply to the data prior to RLE calculation

Details

Uses ggplot2 to create an RLE plot for the selected assay. Any numerical transformation prior to performing the RLE can be specified using the trans argument

Value

A ggplot2 object

Examples

```
data("se")
se$treatment <- c("E2", "E2", "E2", "E2DHT", "E2DHT", "E2DHT")
se$sample <- colnames(se)
## A conventional RLE Plot using all samples
plotAssayRle(se, trans = "log1p", fill = "treatment")
## Calculate RLE within groups
plotAssayRle(se, trans = "log1p", fill = "treatment", rle_group = "treatment")
# Or show groups combined
plotAssayRle(se, trans = "log1p", fill = "treatment", by_x = "treatment")
```

plotGrlCol *Draw a plot from a GRangesList column*

Description

Draw a plot from a GRangesList column using ggplot2

Usage

```
plotGrlCol(
  x,
  var = "width",
  geom = c("boxplot", "violin", "point", "jitter"),
  .id = "sample",
  df,
  fill,
  colour,
  q = 0.1,
  q_size = 3.5,
  qline_type = 2,
  qline_col = "blue",
  total = "{comma(n)}",
  total_geom = c("label", "text", "none"),
  total_pos = c("median", "top", "bottom"),
  total_size = 3.5,
  total_alpha = 1,
  total_adj = 0.025,
  ...,
  digits = 0
)
```

Arguments

x	A GRangesList
var	The variable to plot. Either a column in the mcols element or width. Can be quoted or unquoted
geom	Choose between different geoms, or even provide a geom_*() function
.id	The column name to place the element names. Passed internally to the same argument in bind_rows
df	Optional data.frame with columns to be passed to the colour or fill parameters. Must contain a column with the same name as the value passed to the .id argument.
fill, colour	Optional column names found in the df. Can be quoted or unquoted
q	The overall percentile to be drawn as a labelled, horizontal line. Set q = 0 to hide this line

q_size	Text size of percentile label
qline_type, qline_col	Linetype and colour arguments for the horizontal line showing the specified percentile(s)
total	Glue syntax for totals, representing the length of each GRangesList element
total_geom	Passed to annotate. Set to none to hide totals
total_pos	Position for placing totals
total_size, total_alpha	Size and transparency of totals
total_adj	Adjustment for labels
...	Passed to the geom if selecting via character string. Ignored otherwise
digits	Number of decimal places for the horizontal line label

Details

Using a common column or the width of the ranges, produces a boxplot or violinplot from each element of the provided GRangesList. The names of the GRangesList will be passed to the x-axis using the .id argument. A data frame containing annotations corresponding to each element can be supplied, ensuring that the column associated with each elements is the name passed to the .id argument.

If q is > 0, a horizontal line will be draw corresponding to this percentile across the complete dataset, with parameters for this line able to be set using the qline_* arguments. The digits argument controls how many decimal points will be shown for the associated label.

The total length of each element will be added by default as a total, and is able to be placed across the median values, or at the top and bottom extremes of the plot.

Value

A ggplot object

Examples

```
## Load some peaks
data('peaks')
names(peaks) <- gsub("_peaks.+", "", names(peaks))

## The default boxplot
plotGrlCol(peaks)

## A customised violin plot
df <- data.frame(sample = names(peaks), treat = rep(c("A", "B"), each = 3))
plotGrlCol(
  peaks, geom = "violin", total_pos = "bottom", total_adj = 0.05,
  df = df, fill = "treat",
  draw_quantiles = 0.5, trim = FALSE, width = 0.7, alpha = 0.7
) +
scale_y_log10()
```

```
plotGrlCol(
  peaks, var = score, geom = "jitter", total_pos = "bottom", total_adj = 0.05,
  df = df, colour = treat, width = 0.2, height = 0
)

plotGrlCol(
  peaks, geom = geom_boxplot(colour = "grey70"), df = df, fill = treat,
  total_pos = "bottom", total_adj = 0.05,
) +
scale_y_log10()
```

---

plotHFGC *Plot a Genomic Region showing HiC, Features, Genes and Coverage*

Description

Plot a region with showing HiC, Features, Genes and Coverage

Usage

```
plotHFGC(
  gr,
  hic,
  features,
  genes,
  coverage,
  annotation,
  zoom = 1,
  shift = 0,
  max = 1e+07,
  axistrack = TRUE,
  cytobands,
  covtype = c("l", "heatmap"),
  linecol = c(),
  gradient = grDevices::hcl.colors(101, "viridis"),
  hiccol = list(anchors = "lightblue", interactions = "red"),
  featcol,
  genecol,
  annotcol,
  highlight = "blue",
  hicsize = 1,
  featsize = 1,
  genesize = 1,
  covsize = 4,
  annotsize = 0.5,
  hicname = "HiC",
  featname = "Features",
```

```
  featstack = c("full", "hide", "dense", "squish", "pack"),
  collapseTranscripts = "auto",
  maxTrans = 12,
  ylim = NULL,
  ...,
  fontsize = 12,
  cex.title = 0.8,
  rotation.title = 0,
  col.title = "white",
  background.title = "lightgray",
  title.width = 1.5
)
```

Arguments

gr	The range(s) of interest. Must be on a single chromosome
hic	Any HiC interactions to be included as a GenomicInteractions object. If not supplied, no HiC track will be drawn.
features	A named GRangesList or list of GRangesList objects. Each GRangesList should contain features in each element which will drawn on the same track. If providing a list, each GRangesList within the list will drawn on a separate track. If this argument is not specified, no feature track will be drawn. Features will be drawn with colours provided in `featcol`.
genes	A GRanges object with exon structure for each transcript/gene. If not included, no track will be drawn for gene/transcript structure. The expected mcols in this object are `type`, `gene`, `exon` `transcript` and `symbol`. See data(ex_trans) for an example.
coverage	A named list of BigWigFileList objects containing the primary tracks to show coverage for. Each list element will be drawn on a separate track, with elements within each BigWigFileList shown on the same track. List names will become track names. Alternatively, a single BigWigFileList will plot all individual files on separate tracks. If not included, no coverage tracks will be drawn.
annotation	Annotations for the coverage track(s). A single GRangesList if coverage is a BigWigListList. If coverage is supplied as a list of BigWigFileLists, a named list of GRangesList objects for each coverage track being annotatated. Names must match those given for coverage.
zoom	Multiplicative factor for zooming in and out
shift	Shift the plot. Applied after zooming
max	The maximum width of the plotting region. Given that the width of the final plotting window will be determined by any HiC interactions, this argument excludes any interactions beyond this distance. Plotting can be somewhat slow if any long range interactions are included. Ignored if no HiC interactions are supplied.
axistrack	logical. Add an AxisTrack()
cytobands	Cytogenetic bands to be displayed on each chromosome. See data('grch37.cytobands') for the correct format. Only drawn if a cytobands data.frame is provided.

`covtype` The plot type for coverage. Currently only lines ("l") and heatmaps ("heatmap") are supported

`linecol` If passing a BigWigFileList to coverage, a vector of colours. If passing a list of BigWigFileList objects to coverage, a list of colours with structure that matches the object being passed to coverage, i.e. a named list of the same length, with elements who's length matches each BigWigFileList. Only used if covtype = "l".

`gradient` Colour gradient for heatmaps

`hiccol` list with names `"anchors"` and `"interactions"`. Colours are passed to these elements

`featcol` Named vector (or list) of colours for each feature. Must be provided if drawing features

`genecol` Named vector (or list) of colours for each gene category

`annotcol` Colours matching the coverage annotations

`highlight` Outline colour for the highlight track. Setting this to `NULL` will remove the highlight

`hicsize`, `featsize`, `genesize`, `covsize`, `annotsize`
Relative sizes for each track (hic, features, genes, coverage & annotation)

`hicname`, `featname`
Names displayed in the LHS panel

`featstack` Stacking for the fature track

`collapseTranscripts`
Passed to GeneRegionTrack for the genes track. Defaults to `"auto"` for automatic setting. If the number of transcripts to be plotted is > `maxtrans`, the argument will be automatically set to `"meta"`, otherwise this will be passed as `FALSE` which will show all transcripts.

`maxTrans` Only used if `collapseTranscripts` is set to "auto".

`ylim` If a numeric vector, this will be passed to all coverage tracks. Alternatively, a named list of y-limits for each coverage track with names that match those in each element of the coverage list.

`...` Passed to DataTrack for the **coverage tracks** only. Useful arguments may be things like `legend`

`fontsize` Applied across all tracks

`cex.title` Passed to all tracks

`rotation.title` Passed to all tracks

`col.title` Passed to all tracks

`background.title`
Passed to all tracks

`title.width` Expansion factor passed to plotTracks, and used to widen the panels on the LHS of all tracks. Can have unpredictable effects on the font size of y-axis limits due to the algorithm applied by `plotTracks`

Details

Convenience function for plotting a common set of tracks. All tracks are optional. For more fine control, users are advised to simply use Gviz directly.

The primary tracks defined in this function are H (HiC), F (features), G (genes), and C (coverage). Axis and Ideogram tracks are an additional part of this visualisation, with the Ideogram also being optional

Use all tracks specific to this dataset to generate a simple visualisation. In descending order the tracks displayed will be:

1. HiC Interactions (if supplied)
2. Regulatory features
3. Genes/genes
4. Coverage tracks as supplied

All tracks are optional and will simply be omitted if no data is supplied. See individual sections below for a more detailed explanation of each track

If wanting a single track of genes, simply pass a GRanges object in the format specified for a GeneRegionTrack. Passing a GRangesList with the same format will yield an individual track for each list element, with each track shown by default as a separate colour. This can be used for showing Up/Down-regulated genes, or Detected/Undetected genes.

If passing a BigWigFileList for the coverage track, each file within the object will be drawn on a separate track. If specified, the same y-limits will be applied to each track If passing a list of BigWigFileList objects, each list element will be drawn as a single track with the individual files within each BigWigFileList overlaid within each track.

Cytogenetic band information must be in the structure required by IdeogramTrack, with data for both GRCh37 and GRCh38 provided in this package (grch37.cytobands, grch38.cytobands).

A highlight overlay over the GRanges provided as the `gr` argument will be added if a colour is provided. If set to NULL, no highlight will be added.

Value

A Gviz object

Displaying HiC Interactions

The available arguments for displaying HiC Interactions are defined below. If `hic` is supplied, a single InteractionTrack will be added displaying all interactions with an anchor within the range specified by `gr`. Only interactions with an anchor explicitly overlapping `gr` will be shown. If no interactions are found within `gr`, the track will not be displayed. The **plotting range will expand to incorporate these interactions**, with the paramater `max` providing an upper limit on the displayed range.

hic This is the `GInteractions` object required for inclusion of a HiC track in the final output. Will be ignored if not supplied

hiccol Determines the colours used for display of anchors and interactions

hicsize Relative size of the track compared to others

hicname The name to display on the LHS panel

max The maximum width of the plotted region. If multiple long-range interactions are identified, this provides an upper limit for the display. This defaults to `10Mb`.

Displaying Features

If wanting to add an AnnotationTrack with regions defined as 'features', the following arguments are highly relevant. All are ignored if `features` is not provided.

features A named `GRangesList`. Each element will be considered as a separate feature and drawn as a block in a distinct colour. Any `mcols` data will be ignored.

featcol A **named** vector (or list) providing a colour for each element of `features`

featname The name to display on the LHS panel

featstack Stacking to be applied to all supplied features

featsize Relative size of the track compared to others

Displaying Genes And Transcripts

To display genes or transcripts, simply provide a single `GRanges` object if you wish to display all genes on a single track. The `mcols` element of this object should contain the columns `feature`, `gene`, `exon`, `transcript` and `symbol` as seen on the GeneRegionTrack help page.

Alternatively, a `GRangesList` can be provided to display genes on separate tracks based on their category. This can be useful for separating and colouring Up/Down regulated genes in a precise way. All elements should be as described above. Again, all parameters associated with this track-set will be ignored of no object is supplied to this argument.

genes A `GRanges` or `GRangesList` object as described above

genecol A single colour if supplying a `GRanges` object, or a **named** vector/list of colours matching the `GRangesList`

genesize Relative size of the track compared to others

collapseTranscripts Passed to all tracks. See the GeneRegionTrack section in settings for detail regarding possible arguments. If genes is a `GRangesList`, can be a **named** vector/list with names matching the names of the `genes` object.

Displaying Coverage Tracks

This section contains the most flexibility and can take two types of input. The first option is a `BigWigFileList`, which will lead to each BigWig file being plotted on it's own track. An alternative is a list of `BigWigFileList` objects. In this case, each list element will be plotted as a separate track, with all individual `BigWig` files within each list element overlaid within the relevant track.

In addition to the coverage tracks, annotations can be added to each `BigWigFileList` in the form of coloured ranges, indicating anything of the users choice. Common usage may be to indicate regions with binding of a ChIP target is found to be detected, unchanged, gained or lost.

coverage A `BigWigFileList` or `list` of `BigWigFileList` objects. A single `BigWigFileList` will be displayed with each individual file on a separate track with independent y-axes. Each element of the `BigWigFileList` **must be named** and these names will be displayed on the LHS

panels A list of BigWigFileList objects will be displayed with each list element as a separate track, with any BigWig files overlaid using the same y-axis. The list **must be named** with these names displayed on the LHS panel. Each internal BigWig within a BigWigFileList must also be named.

covtype Currently only lines (covtype = "l") and heatmaps (covtype = "heatmap") are supported. Colours can be specified using the arguments below

linecol Can be a single colour applied to all tracks, or a *named* vector (or list) of colours. If coverage is a single BigWigFileList, these names should match the names of this object exactly. If coverage is a list of BigWigFileList objects, linecol should be a list with matching names. Each element of this list should also be a **named** vector with names that exactly match those of each corresponding BigWigFileList.

gradient A colour gradient applied to all heatmap tracks. No specific structure is required beyond a vector of colours.

covsize Relative size of the tracks compared to others

ylim Can be a vector of length 2 applied to all coverage tracks. Alternatively, if passing a list of BigWigFlieList objects to the coverage argument, this can be a **named** list of numeric vectors with names matching coverage

annotation Each BigWigFileList needs annotations to be passed to this argument as a **named** GRangesList, with names being used to associate unique colours with that set of ranges. If coverage is a BigWigFileList a simple GRangesList would be supplied and a single 'annotation' track will appear at the top of the set of coverage tracks. If coverage is a list, then a **named** list of GRangesList objects should be supplied, with each being displayed above the corresponding track from the coverage object.

annotcol A vector of colours corresponding to all names within all GRangesList elements supplied as annotation. It is assumed that the same colour scheme will be applied to all annotation tracks and, as such, the colours should **not** be provided as a list which matches the coverage tracks. Instead, every named element anywhere in the annotation GRanges, across all of the tracks must be included as a colour

annotsize Relative size of the tracks compared to others

Examples

```
library(rtracklayer)
## Make sure we have the cytobands active
data(grch37.cytobands)

## Prepare the HiC, promoter & transcript information
data(ex_hic, ex_trans, ex_prom)
ex_features <- GRangesList(Promoter = ex_prom)
featcol <- c(Promoter = "red")

## Prepare the coverage
fl <- system.file(
"extdata", "bigwig", c("ex1.bw", "ex2.bw"), package = "extraChIPs"
)
bwfl <- BigWigFileList(fl)
names(bwfl)  <- c("ex1", "ex2")
bw_col <- c(ex1 = "#4B0055", ex2 = "#007094")
```

```
## Define the plotting range
gr <- GRanges("chr10:103862000-103900000")

## Now create the basic plot
plotHFGC(
  gr,
  hic = ex_hic, features = ex_features, genes = ex_trans, coverage = bwfl,
  featcol = featcol, linecol = bw_col, cytobands = grch37.cytobands
)

plotHFGC(
  gr,
  hic = ex_hic, features = ex_features, genes = ex_trans, coverage = bwfl,
  featcol = featcol, linecol = bw_col, cytobands = grch37.cytobands,
  maxTrans = 1
)
```

---

plotOverlaps *Plot Overlaps Between List Elements*

Description

Plot Overlaps between list elements as an upset or Venn diagram

Usage

```
plotOverlaps(x, ...)

## S4 method for signature 'GRangesList'
plotOverlaps(
  x,
  type = c("auto", "venn", "upset"),
  var = NULL,
  f = c("mean", "median", "max", "min", "sd"),
  merge_within = 1L,
  ignore.strand = TRUE,
  set_col = NULL,
  ...,
  label_size = 3.5,
  hj_sets = 1.15,
  exp_sets = 0.2,
  vj_intersect = -0.5,
  exp_intersect = 0.1
)
```

```
## S4 method for signature 'list'
plotOverlaps(
  x,
  type = c("auto", "venn", "upset"),
  set_col = NULL,
  ...,
  label_size = 3.5,
  hj_sets = 1.15,
  exp_sets = 0.2,
  vj_intersect = -0.5,
  exp_intersect = 0.1
)
```

Arguments

x	GRangesList of S3 list able to be coerced to character vectors
...	Passed to draw.pairwise.venn (or draw.single/triple.venn) for Venn Diagrams, and to simpleUpSet for UpSet plots
type	The type of plot to be produced
var	Column to summarised as a boxplot in an upper panel (UpSet plot only)
f	Summarisation function. Must return a single value from any numeric vector
merge_within	Passed to makeConsensus
ignore.strand	Passed to reduce
set_col	Colours to be assigned to each set
label_size	Text size for set and intersection labels. Passed internally to geom_text(size = label_size)
hj_sets	Horizontal adjustment of set size labels
exp_sets	X-axis expansion for set size panel
vj_intersect	Vertical adjustment of intersection size labels
exp_intersect	Y-axis expansion for intersections size panel

Details

This function should give the capability to show overlaps for any number of replicates or groups, or a list of items such as gene names. For n = 2, a scaled Venn Diagram will be produced, however no scaling is implemented for n = 3

UpSet plots are possible for any lists with length > 1, and are the only implemented possibility for lists > 3.

If the input is a GRangesList an additional boxplot can be requested using any numeric column within the existing mcols() element. Values will be summarised across all elements using the requested function and the boxplot will be included as an upper panel above the intersections

Value

Either a VennDiagram (i.e. grid) object, or a ComplexUpset plot

Examples

```
## Examples using a list of character vectors
ex <- list(
  x = letters[1:5], y = letters[c(6:15, 26)], z = letters[c(2, 10:25)]
)
plotOverlaps(ex, type = "upset")
plotOverlaps(ex, type = "venn", set_col = 1:3, alpha = 0.3)
plotOverlaps(ex[1:2])

## GRangesList object will produce a boxplot of summarised values in the
## upper panel
data("peaks")
grl <- peaks[1:3]
names(grl) <- gsub("_peaks.+", "", names(grl))
plotOverlaps(grl, type = 'upset', var = 'score', f = 'max')

## If only two samples are present, a VennDiagram will be produced
plotOverlaps(grl[1:2], set_col = c("green", "blue"), cex = 1.5, cat.cex = 1.5)
```

---

plotPairwise *Plot Pairwise Values from a GRangeList*

Description

Plot Pairwise Values from a GRangeList by overlapping GRanges

Usage

```
plotPairwise(
  x,
  var,
  colour,
  label,
  index = c(1, 2),
  p = 0,
  method = "union",
  ignore.strand = TRUE,
  min_width = 0,
  xside = c("boxplot", "density", "violin", "none"),
  yside = c("boxplot", "density", "violin", "none"),
  side_panel_width = c(0.3, 0.4),
  side_alpha = 1,
  xside_axis_pos = "right",
  yside_axis_label = scales::label_wrap(10),
  smooth = TRUE,
  rho_geom = c("text", "label", "none"),
  rho_col = "black",
```

```
  rho_size = 4,
  rho_pos = c(0.05, 0.95),
  rho_alpha = 1,
  label_geom = c("label_repel", "label", "text_repel", "text", "none"),
  label_width = 20,
  label_sep = "; ",
  label_size = 3.5,
  label_alpha = 0.7,
  min_d = 1,
  group_sep = " - ",
  simplify_equal = TRUE,
  name_sep = " ",
  plot_theme = theme_get(),
  ...
)
```

Arguments

x	A GRangesList
var	The colunm to compare between list elements
colour	Optional column to use for combining across elements and setting point colour
label	Optional column to use for labelling ranges with the most extreme changes
index	Which list elements to compare
p, method, ignore.strand, min_width	Passed to makeConsensus()
xside, yside	Will call geom_(x/y)side* from the package ggside and show additional panels on the top and right of the plot respectively
side_panel_width	Set the relative widths of the side panels
side_alpha	Set fill transparency in side panels
xside_axis_pos	Position for axis_labels in the top panel when using a discrete axis
yside_axis_label	Wrapping for axis labels on the right-side panel when using a discrete axis. Set to waiver() to turn off wrapping
smooth	logical(1). If TRUE a regression line will be drawn using geom_smooth. To add this manually, set to FALSE and call this geom with any custom parameters after creating the plot
rho_geom	Used to add correlation coefficients for the two values
rho_col, rho_size, rho_alpha	Parameters for displaying the correlation
rho_pos	Place the correlation coefficient within the plotting region
label_geom	Used to add labels from the column specified in label
label_width	Label text will be truncated to this length

label_sep	If multiple values (e.g. genes) are mapped to a range, separate values using this string
label_size, label_alpha	Passed to the geom used for adding labels
min_d	Labels will only be added if the points lie circle beyond a sircle of this radius
group_sep	Text separator used to separate categories when specifying colour
simplify_equal	logical(1) When combining columns from both elements for the colour categories, should shared values be annotated as 'Both ...' instead of having longer, more difficult to read annotations.
name_sep	Character string to separate names of the GRangesList and the selected column. Will appear as axis-labels
plot_theme	Sets the initial theme by using the default theme for the current R session via get_theme()
...	Passed to geom_point() for the main panel

Details

This function enables pairwise plotting of two elements within a GRangesList. All elements of the GRangesList will contain the same columns, so a set of consensus ranges are first formed, before then taking all values from each GRangesList element which overlap the range and producing a piarwise plot.

Given that not all ranges will have values in both elements, side panels are produced which can show the distribution of plotted values, along with those which are only found in one of the foundational GRanges. These can take the form of density, violin or boxplots.

Addition columns, such as Differential Signal status can also be used to form pairwise groups and colour the points.

If a column in the GRangesList is suitable for labelling points, such as a column with genes mapped to each range, this can be specified using the argument `label = "col_to_label"`. Only the furthest point from the origin will be labelled within each group used to colour the points. Labels will only be added if they lie beyond a circle of radius `min_d` from the origin. If multiple genes are mapped to the range, these will be separated by the string provided in the `label_sep` argument.

A regression line and correlation co-efficient are added to the plot by default, but can be hidden easily if preferred

Value

A ggside or ggplot2 object

Examples

```
theme_set(theme_bw())
set.seed(100)
gr1 <- GRanges(paste0("chr1:", seq(10, 150, by = 10)))
width(gr1) <- 5
gr1$logFC <- rnorm(length(gr1))
gr1$FDR <- rbeta(length(gr1), 1, 8)
```

```
gr2 <-  GRanges(paste0("chr1:", seq(51, 250, by = 15)))
width(gr2) <- 4
gr2$logFC <- rnorm(length(gr2))
gr2$FDR <- rbeta(length(gr2), 1, 8)
grl <- GRangesList(TF1 = gr1, TF2 = gr2)
grl <- addDiffStatus(grl)

# Using the defaults
plotPairwise(grl, var = "logFC")

# Density plots on the side panels
plotPairwise(
  grl, var = "logFC", xside = "density", yside = "density",
  side_alpha = 0.5
)

# Turning off side panels, regression line and correlations
plotPairwise(
  grl, var = "logFC", xside = "none", yside = "none",
  rho_geom = "none", smooth = FALSE
)

# Add colours using the status column
plotPairwise(grl, var = "logFC", colour = "status") +
  scale_fill_manual(values = rep_len(c("blue", "red", "white", "grey"), 8)) +
  guides(fill = "none")
```

---

plotPie *Draw Pie Graphs based on one or more columns*

Description

Draw Pie Graphs based one or more data.frame columns

Usage

```
plotPie(object, ...)

## S4 method for signature 'GRanges'
plotPie(object, scale_by = c("n", "width"), ...)

## S4 method for signature 'DataFrame'
plotPie(object, ...)

## S4 method for signature 'data.frame'
plotPie(
  object,
```

```
  fill,
  x,
  y,
  scale_by,
  scale_factor = 1000,
  width = 0.8,
  total_geom = c("label", "text", "none"),
  total_glue = "{comma(N)}",
  total_colour = "black",
  total_fill = "white",
  total_alpha = 1,
  total_size = 3,
  min_p = 0.01,
  max_p = 1,
  cat_geom = c("label", "text", "none"),
  cat_glue = "{.data[[fill]]}\n{comma(n, 1)}\n({percent(p, 0.1)})",
  cat_colour = "black",
  cat_fill = "white",
  cat_size = 3,
  cat_alpha = 1,
  cat_adj = 0,
  hole_width = 0,
  ...
)
```

Arguments

`object`	An object (`data.frame`)
`...`	Not used
`scale_by`	Scale the counts by this column. In this case of a GRanges object this defaults to the count (scale_by = "n") but can also be specified as being width of each range (scale_by = "width"). If choosing width, width will be displayed in Kb
`fill`	The category/column used to fill the slices of the pie charts
`x`	The second (optional) category/column to place along the x-axis
`y`	The final (optional) category/column to plce along the y-axis
`scale_factor`	When scaling by another column, such as width, totals will be divided by this value, with 1000 being the default to provide output in kb.
`width`	Scale the width of all pies
`total_geom`	The geom_* to use for the totals at the centre of each pie. Setting this to 'none' will disable totals
`total_glue`	glue syntax to use for the totals in the centre of each pie. The column 'N' will produce the totals and any other values or formatting may be added here.
`total_colour`, `total_fill`, `total_alpha`, `total_size`	Colour, fill, alpha and size for the main totals in the centre of each pie chart

min_p	The minimum proportion of the total required for adding labels. Effectively removes labels from pie charts with few members. Alternatively when only one column is specified, categories below this will not be shown around the edge of the plot
max_p	only display labels for segments representing less than this proportion of the total.
cat_geom	The geom_* to use for category labels corresponding to each slice of the pie. Setting this to 'none' will disable category labels
cat_glue	glue syntax to use for the category labels corresponding to each slice of the pie charts. The columns 'n' and 'p' can be used to print totals and proportions for each slice.
cat_colour, cat_fill, cat_size, cat_alpha	Colour, fill, size and alpha for category labels
cat_adj	Adjust category labels
hole_width	Add a hole in the middle to turn the plot into a donut. Values between zero and 1 work best. Only implemented for pie charts using one value (i.e. fill)

Details

Using a data.frame as input, this function will draw pie graphs based on one ore more columns, by simply counting the values in combination across these columns. One column must be selected for the fill as a bare minimum, with up to three being possible. Additional columns can be set for the x-axis to draw a series of pie-graphs in a row, with a further column able to added to layout a series of pie graphs in a grid

If only one column/category is chosen, category labels will be added around the edge of the plot

If show_total = TRUE the overall counts for each pie graph will be added in the centre using geom_label. Parameters for these labels are customisable

Value

A ggplot2 object able to be customised with colour scales and themes.

Also note that the $data element of the returned object will contain the data.frame used for plotting. The additional column label_radians represents the mid-point of each pie slice and can be used for manually adding labels to each pie. Only applies when plotting across the x or y axes

Examples

```
set.seed(200)
df <- data.frame(
  feature = sample(
    c("Promoter", "Enhancer", "Intergenic"), 200, replace = TRUE
  ),
  TF1 = sample(c("Up", "Down", "Unchanged"), 200, replace = TRUE),
  TF2 = sample(c("Up", "Down", "Unchanged"), 200, replace = TRUE),
  w = rexp(200)
)
plotPie(df, fill = "feature", total_glue = "N = {comma(N)}")
plotPie(
```

```
  df, fill = "feature", scale_by = "w", total_geom = "none",
  cat_glue = "{percent(p)}", cat_size = 5
)
plotPie(df, fill = "feature", x = "TF1")
plotPie(
  df, fill = "feature", x = "TF1", y = "TF2", min_p = 0.02,
  total_geom = "none", cat_glue = "{n} / {N}"
 ) +
 scale_fill_viridis_d() +
 theme_bw()


## And using a GRanges object
data("ex_prom")
gr <- ex_prom
mcols(gr) <- df[seq_along(gr),]
## Show values by counts
plotPie(gr, fill = "feature", total_size = 5)
## Show values scaled by width of each range as a donut plot
plotPie(
  gr, fill = "feature", scale_by = "width", total_glue = "{round(N, 1)}kb",
  cat_glue = "{percent(p, 0.1)}", cat_size = 4, total_size = 5, hole_width = 0.2
)
```

---

plotProfileHeatmap *Draw a coverage Profile Heatmap*

Description

Plot a coverage Profile Heatmap across multiple ranges

Usage

```
plotProfileHeatmap(object, ...)

## S4 method for signature 'GenomicRangesList'
plotProfileHeatmap(
  object,
  profileCol = "profile_data",
  xValue = "bp",
  fillValue = "score",
  facetX = NULL,
  facetY = NULL,
  colour = facetY,
  linetype = NULL,
  summariseBy = c("mean", "median", "min", "max", "none"),
  xLab = xValue,
  yLab = NULL,
```

```
  fillLab = fillValue,
  labelFunX = waiver(),
  relHeight = 0.3,
  sortFilter = NULL,
  maxDist = 100,
  ...
)

## S4 method for signature 'GenomicRanges'
plotProfileHeatmap(
  object,
  profileCol = "profile_data",
  xValue = "bp",
  fillValue = "score",
  facetX = NULL,
  facetY = NULL,
  colour = facetY,
  linetype = NULL,
  summariseBy = c("mean", "median", "min", "max", "none"),
  xLab = xValue,
  yLab = NULL,
  fillLab = fillValue,
  labelFunX = waiver(),
  relHeight = 0.3,
  summaryLabelSide = "left",
  respectLevels = FALSE,
  sortFilter = NULL,
  maxDist = 100,
  ...
)
```

Arguments

`object`	A GRanges or GRangesList object
`...`	Passed to facet_grid internally. Can be utilised for switching panel strips or passing a labeller function
`profileCol`	Column name specifying where to find the profile DataFrames
`xValue`, `fillValue`	Columns within the profile DataFrames for heatmaps
`facetX`, `facetY`	Columns used for faceting across the x- or y-axis respectively
`colour`	Column used for colouring lines in the summary panel. Defaults to any column used for facetY
`linetype`	Column used for linetypes in the summary panel
`summariseBy`	Function for creating the summary plot in the top panel. If set to 'none', no summary plot will be drawn. Otherwise the top panel will contain a line-plot representing this summary value for each x-axis bin

`xLab`, `yLab`, `fillLab`
: Labels for plotting aesthetics. Can be overwritten using labs() on any returned object

`labelFunX`
: Function for formatting x-axis labels

`relHeight`
: The relative height of the top summary panel. Represents the fraction of the plotting area taken up by the summary panel.

`sortFilter`
: If calling on a GRangesList, a method for subsetting the original object (e.g. 1:2). If calling on a GRanges object should be and expression able to be parsed as a filtering expression using eval_tidy. This is applied when sorting the range order down the heatmap such that ranges can be sorted by one or specific samples, or all. Ranges will always be sorted such that those with the strongest signal are at the top of the plot

`maxDist`
: Maximum distance from the centre to find the strongest signal when arranging the ranges

`summaryLabelSide`
: Side to place y-axis for the summary plot in the top panel

`respectLevels`
: logical(1) If FALSE, facets along the y-axis will be arranged in descending order of signal, otherwise any original factor levels will be retained

Details

Convenience function for plotting coverage heatmaps across a common set of ranges, shared between one or more samples. These are most commonly the coverage values from merged samples within a treatment group. THe input data structure is based on that obtained from getProfileData, and can be provided either as a GRanges object (generally for one sample) or as a GRangesList.

A 'profile DataFrame' here refers to a data.frame (or tibble, or DataFrame) with a coverage value in one column that corresponds to a genomic bin of a fixed size denoted in another, as generated by getProfileData. Given that multiple ranges are most likely to be drawn, each profile data frame must be the same size in terms of the number of bins, each of which represent a fixed number of nucleotides. At a minimum this is a two column data frame although getProfileData will provide three columns for each specified genomic region.

If using a GRangesList, each list element will be drawn as a separate panel by default. These panels will appear in the same order as the list elements of the GRangesList, although this can easily be overwritten by passing a column name to the facetX argument. The default approach will add the original element names as the column "name" which can be seen in the $data element of any resultant ggplot object produced by this function.

Value

A `ggplot2` object, able to be customised using standard `ggplot2` syntax

Examples

```
library(rtracklayer)
fl <- system.file(
"extdata", "bigwig", c("ex1.bw", "ex2.bw"), package = "extraChIPs"
)
```

```
bwfl <- BigWigFileList(fl)
names(bwfl) <- c("ex1", "ex2")

gr <- GRanges(
  c(
    "chr10:103880281-103880460", "chr10:103892581-103892760",
    "chr10:103877281-103877460"
   )
)
pd <- getProfileData(bwfl, gr)
plotProfileHeatmap(pd, "profile_data") +
  scale_fill_viridis_c(option = "inferno", direction = -1) +
  labs(fill = "Coverage")
```

---

plotSplitDonut *Draw Two-Level Donut Charts*

Description

Create Donut charts based on one or two columns in a data frame

Usage

```
plotSplitDonut(object, ...)

## S4 method for signature 'GRanges'
plotSplitDonut(object, scale_by = c("n", "width"), ...)

## S4 method for signature 'DataFrame'
plotSplitDonut(object, ...)

## S4 method for signature 'data.frame'
plotSplitDonut(
  object,
  inner,
  outer,
  scale_by,
  scale_factor = 1000,
  r_centre = 0.5,
  r_inner = 1,
  r_outer = 1,
  total_glue = "{comma(N)}",
  total_size = 5,
  total_colour = "black",
  inner_glue = "{inner} {.data[[inner]]}\n{percent(p,0.1)}",
  outer_glue = "{outer} {.data[[outer]]}\n{percent(p,0.1)}",
```

```
  total_label = c("label", "text", "none"),
  inner_label = c("label", "text", "none"),
  outer_label = c("label", "text", "none"),
  label_alpha = 1,
  inner_label_alpha = NULL,
  outer_label_alpha = NULL,
  label_size = 3,
  inner_label_size = NULL,
  outer_label_size = NULL,
  label_colour = "black",
  inner_label_colour = NULL,
  outer_label_colour = NULL,
  label_fill = "white",
  inner_label_fill = NULL,
  outer_label_fill = NULL,
  min_p = 0.05,
  inner_min_p = NULL,
  outer_min_p = NULL,
  max_p = 1,
  inner_max_p = NULL,
  outer_max_p = NULL,
  inner_pattern = ".",
  outer_pattern = ".",
  inner_rotate = FALSE,
  outer_rotate = FALSE,
  explode_inner = NULL,
  explode_outer = NULL,
  explode_query = c("AND", "OR"),
  explode_x = 0,
  explode_y = 0,
  explode_r = 0,
  nudge_r = 0.5,
  inner_nudge_r = NULL,
  outer_nudge_r = NULL,
  expand = 0.1,
  inner_palette = NULL,
  outer_palette = NULL,
  inner_legend = TRUE,
  outer_legend = TRUE,
  outer_p_by = c("all", "inner"),
  layout = c(main = area(1, 1, 12, 12), lg1 = area(2, 12), lg2 = area(11, 12)),
  ...
)
```

Arguments

object	A GRanges or data.frame-like object
...	Not used

`scale_by` Column to scale values by. If provided, values in this column will be summed, instead of simply counting entries. Any label in the centre of the plot will also reflect this difference

`inner` Column name to create the inner ring

`outer` Column name to create the outer ring, subset by the inner ring

`scale_factor` When scaling by another column, such as width, totals will be divided by this value, with 1000 being the default to provide output in kb.

`r_centre` The radius of the hole in the centre. Setting to zero will create a Pie chart

`r_inner`, `r_outer`
The radii of the inner/outer rings

`total_glue` glue-syntax for formatting the total which appears in the centre of the plot. Internally, the value `N` will be calculated and as such, this value should appear within this argument.

`total_size` Label size total number of entries in the centre of the plot.

`total_colour` Label colour for the summary total in the centre

`inner_glue`, `outer_glue`
glue-syntax for formatting labels which appear on each inner/outer segment Internally, the values `n` and `p` will be calculated as totals and proportions of the total. As such, these values can appear within this argument, as well as the fields described in the details

`total_label`, `inner_label`, `outer_label`
Can take values 'text', 'label' or 'none'. If setting one the first two values, the labelling function `geom_*` will be called, otherwise no label will be drawn

`label_alpha`, `inner_label_alpha`, `outer_label_alpha`
transparency for labels

`label_size`, `inner_label_size`, `outer_label_size`
Size of all text labels

`label_colour`, `inner_label_colour`, `outer_label_colour`
Takes any colour specification, with the additional option of 'palette'. In this special case, the same palette as is used for each segment will be applied.

`label_fill`, `inner_label_fill`, `outer_label_fill`
Takes any fixed colour

`min_p`, `inner_min_p`, `outer_min_p`
only display labels for segments representing greater than this proportion of the total. If inner/outer values are specified, the values in `min_p` will be ignored for that layer

`max_p`, `inner_max_p`, `outer_max_p`
only display labels for segments representing less than this proportion of the total. If inner/outer values are specified, the values in `max_p` will be ignored for that layer

`inner_pattern`, `outer_pattern`
Regular expressions which are combined with max_p and min_p values for accurately choosing labels

`inner_rotate`, `outer_rotate`
: logical(1). Rotate labels for inner or outer rings. This will be ignored by when setting the geom as "label". See geom_text

`explode_inner`, `explode_outer`
: Regular expressions from either the inner or outer ring for which segments will be 'exploded'

`explode_query`
: Setting to AND and specifying values for both the inner and outer ring will require matches in both categories

`explode_x`, `explode_y`
: Numeric values for shifting exploded values

`explode_r`
: Radius expansion for exploded values

`nudge_r`, `inner_nudge_r`, `outer_nudge_r`
: Radius expansion for labels

`expand`
: Passed to expansion for both x and y axes. Can be helpful if labels are clipped by plot limits

`inner_palette`
: Colour palette for the inner ring

`outer_palette`
: Optional colour palette for the outer ring

`inner_legend`, `outer_legend`
: logical(1). Show legends for either layer

`outer_p_by`
: Scale the proportions for outer segments by the complete dataset, or within each inner segment

`layout`
: Passed to plot_layout

Details

Using a data.frame or GRanges object, this function enables creation of a Pie/Donut chart with an inner and outer ring. The function itself is extremely flexible allowing for separate colour palettes in the inner and outer rings, as well as highly customisable labels.

Sections can be exploded using a value from the inner ring or outer ring separately, or in combination by setting `explode_query = "AND"`. Exploded sections can be shifted by expanding the radius (`explode_r`), or along the x/y co-ordinates using `explode_x/y`, allowing for detailed placement of sections.

If only the inner palette is specified, segments in the outer ring will be assigned the same colours as the inner segments, but with increased transparency. Only a single legend will be drawn in this scenario. If an outer palette is specified, both colour palettes are completely distinct and two distinct legends will be drawn. The placement of these legends, along with the larger donut plot, can be manually specified by providing a layout as defined in plot_layout. Names are not required on this layout, but may be beneficial for code reproducibility.

The inner label denoting the total can also be heavily customised using the glue syntax to present the calculated value `N` along with any additional text, such as 'kb' if scaling GenomicRanges by width. The same approach can be taken for the inner and outer labels, where totals are held in the value `n`, proportions are held in the value `p` and the values corresponding to each segment can be accessed using `.data[[inner]]` or `.data[[outer]]`. Column titles can be added using `{inner}`/`{outer}`. Values from the inner segments can be added to the outer labels using this strategy enabling a wide variety of labelling approaches to be utilised.

Value

A patchwork object consisting of both ggplot2 objects and legend grobs

Examples

```
library(grDevices)
set.seed(200)
df <- data.frame(
  feature = sample(
    c("Promoter", "Enhancer", "Intergenic"), 200, replace = TRUE
  ),
  TF1 = sample(c("Up", "Down", "Unchanged"), 200, replace = TRUE),
  TF2 = sample(c("Up", "Down", "Unchanged"), 200, replace = TRUE)
)
## The standard plot
plotSplitDonut(df, inner = "TF1", outer = "TF2", inner_legend = FALSE)

## Adding an exploded section along with an outer palette & customisation
plotSplitDonut(
  df, inner = "TF1", outer = "feature", total_label = "none",
  inner_label_alpha = 0.5, r_centre = 0,
  outer_glue = "{.data[[outer]]}\n(n = {n})", outer_label = "text",
  explode_inner = "Up", explode_outer = "Prom|Enh",
  explode_query = "AND", explode_r = 0.4, outer_rotate = TRUE,
  inner_palette = hcl.colors(3, "Spectral", rev = TRUE),
  outer_palette = hcl.colors(3, "Cividis")
)
```

---

propOverlap *Find the proportions of an overlapping range*

Description

Find the proportion of a query reange which overlaps the subject

Usage

```
propOverlap(x, y, ...)

## S4 method for signature 'GRanges,GRanges'
propOverlap(x, y, ignore.strand = FALSE, ...)
```

Arguments

x, y	A GenomicRanges object
...	Not used
ignore.strand	If set to TRUE, then the strand of x and y is set to "*" prior to any computation.

Details

This behaves similarly to overlapsAny except the proportion of the query range which overlaps one or more subject ranges is returned instead of a logical vector

Value

Numeric vector the same length as x

Examples

```
x <- GRanges("chr1:1-10")
y <- GRanges("chr1:1-5")
propOverlap(x, y)
propOverlap(y, x)
```

---

reduceMC *Reduce ranges retaining mcols*

Description

Reduce ranges retaining mcols

Usage

```
reduceMC(x, ignore.strand = FALSE, simplify = TRUE, ...)
```

Arguments

x	A GenomicRanges object
ignore.strand	If set to TRUE, then the strand of x and y is set to "*" prior to any computation.
simplify	logical(1). Attempt to simplify returned columns where possible
...	Passed to reduce

Details

This function extends reduce so that **all** `mcols` are returned in the output. Where the reduced ranges map to multiple ranges in the original range, `mcols` will be returned as `CompressedList` columns.

If `simplify = TRUE` columns will be returned as vectors where possible.

Value

A GRanges object

Examples

```
x <- GRanges(c("chr1:1-10:+", "chr1:6-12:-"))
x$id <- c("range1", "range2")
reduceMC(x)
reduceMC(x, ignore.strand = TRUE)
```

`setoptsMC` *Perform set operations retaining mcols*

Description

Perform set operations retaining all mcols from the query range

Usage

```
setdiffMC(x, y, ...)

intersectMC(x, y, ...)

unionMC(x, y, ...)

## S4 method for signature 'GRanges,GRanges'
setdiffMC(x, y, ignore.strand = FALSE, simplify = TRUE, ...)

## S4 method for signature 'GRanges,GRanges'
intersectMC(x, y, ignore.strand = FALSE, simplify = TRUE, ...)

## S4 method for signature 'GRanges,GRanges'
unionMC(x, y, ignore.strand = FALSE, simplify = TRUE, ...)
```

Arguments

x, y	GenomicRanges objects
...	Not used
ignore.strand	If set to TRUE, then the strand of x and y is set to "*" prior to any computation.
simplify	logical(1) If TRUE, any List columns will be returned as vectors where possible. This can only occur if single, unique entries are present in all initial elements.

Details

This extends the methods provided by setdiff, intersect and union so that `mcols` from `x` will be returned as part of the output.

Where output ranges map back to multiple ranges in `x`, `CompressedList` columns will be returned. By default, these will be simplified if possible, however this behaviour can be disabled by setting `simplify = FALSE`.

All columns will be returned which can also be time-consuming. A wise approach is to only provide columns you require as part of the query ranges `x`.

If more nuanced approaches are required, the returned columns can be further modified by many functions included in the `plyranges` package, such as `mutate()`.

Value

A GRanges object with all mcols returned form the original object. If a range obtained by setdiff maps back to two or more ranges in the original set of Ranges, mcols will be returned as CompressedList columns

Examples

```
x <- GRanges("chr1:1-100:+")
x$id <- "range1"
y <- GRanges(c("chr1:51-60:+", "chr1:21-30:-"))
setdiffMC(x, y)
setdiffMC(x, y, ignore.strand = TRUE)

# The intersection works similarly
intersectMC(x, y)

# Union may contain ranges not initially in x
unionMC(x, y)
unionMC(x, y, ignore.strand = TRUE)
```

---

stitchRanges *Stitch Ranges within a given distance*

Description

Stitch together ranges within a given distance, using excluded ranges as barriers that cannot be crossed

Usage

```
stitchRanges(x, exclude, maxgap = 12500L, ignore.strand = TRUE)
```

Arguments

`x`	Ranges to be stitched together
`exclude`	Ranges to exclude
`maxgap`	The maximum distance between ranges to be stitched
`ignore.strand`	logical

Details

Stitches together ranges within a given distance, using any ranges provided for exclusion as barriers between stitched ranges. This may be particularly useful if wanting to stitch enhancers whilst excluding promoters.

All inputs and outputs are Genomic Ranges objects

Value

A GRanges object

Examples

```
x <- GRanges(c("chr1:1-10", "chr1:101-110", "chr1:201-210", "chr2:1-10"))
y <- GRanges("chr1:200:+")
stitchRanges(x, exclude = y, maxgap = 100)
```

---

voomWeightsFromCPM *Estimate voom precision weights directly From CPM values*

Description

Estimate voom precision weights directly From CPM values

Usage

```
voomWeightsFromCPM(
  cpm,
  design = NULL,
  w0 = NULL,
  lib.size = NULL,
  isLogCPM = TRUE,
  span = 0.5,
  ...
)
```

Arguments

cpm	Matrix of CPM or logCPM values
design	The design matrix for the experiment
w0	Initial vector of sample weights. Should be calculated using arrayWeights
lib.size	Initial library sizes. Must be provided as these are no estimable from CPM values
isLogCPM	logical(1). Indicates whether the data is log2 transformed already. Most commonly (e.g. if using the output of cqn) it will be,
span	Width of the smoothing window used for the lowess mean-variance trend. Expressed as a proportion between 0 and 1.
...	Passed to lmFit internally

Details

This function takes CPM or logCPM values and estimates the precision weights as would be done by providing counts directly to voom. Using this function enables the use of logCPM values which have been normalised using other methods such as Conditional-Quantile or Smooth-Quantile Normalisation.

The precision weights are returned as part of the `EList` output, and these are automatically passed to the function lmFit during model fitting. This will ensure that the mean-variance relationship is appropriate for the linear modelling steps as performed by limma.

Initial sample weights can be passed to the function, and should be calculated using arrayWeights called on the normalised logCPM values. The returned sample weights will be different to these, given that the function voomWithQualityWeights performs two rounds of estimation. The first is on the initial data, with the inappropriate mean-variance relationship, whilst the second round is after incorporation of the precision weights.

Value

An object of class `EList` as would be output by voom. Importantly, there will be no `genes` element, although this can be added later. Similarly, the returned `targets` element will only contain sample names and library sizes. This can be incorporated with any other metadata as required.

Plotting data is always returned, noting the the value `sx` has been offset by the library sizes and will be simple logCPM values. As such, the fitted `Amean` is also returned in this list element.

If initial sample weights were provided, modified weights will also be returned, as the initial function voomWithQualityWeights performs two rounds of estimation of sample weights. Here we would simply provide the initial weights a priori, with the second round performed within the function. Importantly, this second round of sample weight estimation uses the precision weights ensuring the correct mean-variance relationship is used for the final estimation of sample weights

Examples

```
library(csaw)
library(edgeR)
bamFiles <- system.file("exdata", c("rep1.bam", "rep2.bam"), package="csaw")
wc <- windowCounts(bamFiles, filter=1)
cpm <- cpm(wc, log = TRUE)
el <- voomWeightsFromCPM(cpm, lib.size = wc$totals)
```

Index

∗ **datasets**
    cytobands, 16
    ex_datasets, 23
    fixed_width_datasets, 26
∗ **internal**
    .makeFinalProfileHeatmap, 4
    .mapFeatures, 5
    .mapGi, 6
    .mapWithin, 7
    extraChIPs-package, 3
.makeFinalProfileHeatmap, 4
.mapFeatures, 5
.mapGi, 6
.mapWithin, 7

addDiffStatus, 7
addDiffStatus,data.frame-method
        *(*addDiffStatus*)*, 7
addDiffStatus,DataFrame-method
        *(*addDiffStatus*)*, 7
addDiffStatus,GRanges-method
        *(*addDiffStatus*)*, 7
addDiffStatus,GRangesList-method
        *(*addDiffStatus*)*, 7
addDiffStatus,SummarizedExperiment-method
        *(*addDiffStatus*)*, 7
annotate, *52*
AnnotationTrack, *57*
arrayWeights, *78, 79*
as_tibble, 9
as_tibble(), *3*

bestOverlap, 10
bestOverlap(), *3*
bestOverlap,GRanges,GRanges-method
        *(*bestOverlap*)*, 10
bestOverlap,GRanges,GRangesList-method
        *(*bestOverlap*)*, 10
bind_rows, *51*
bplapply, *28*

centrePeaks, 12
centrePeaks,GRanges,BamFile-method
        *(*centrePeaks*)*, 12
centrePeaks,GRanges,BamFileList-method
        *(*centrePeaks*)*, 12
centrePeaks,GRanges,BigWigFile-method
        *(*centrePeaks*)*, 12
centrePeaks,GRanges,BigWigFileList-method
        *(*centrePeaks*)*, 12
centrePeaks,GRanges,character-method
        *(*centrePeaks*)*, 12
chop, *14*
chopMC, 13
chopMC(), *3*
cluster-direction, *39*
collapseGenes, 14
collapseGenes(), *4*
colToRanges, 15
colToRanges(), *3*
colToRanges,data.frame-method
        *(*colToRanges*)*, 15
colToRanges,DataFrame-method
        *(*colToRanges*)*, 15
colToRanges,GRanges-method
        *(*colToRanges*)*, 15
combineTests, *43*
CompressedList, *77*
cytobands, 16

DataTrack, *55*
defineRegions, 17
defineSeqinfo, 19
distinct, *20*
distinctMC, 20
distinctMC(), *3*
draw.pairwise.venn, *60*
dualFilter, 21
dualFilter(), *3*

eBayes, *25, 26*

eval_tidy, *69*
ex_datasets, 23
ex_genes *(*ex_datasets*)*, 23
ex_hic *(*ex_datasets*)*, 23
ex_prom *(*ex_datasets*)*, 23
ex_trans *(*ex_datasets*)*, 23
expansion, *73*
extraChIPs *(*extraChIPs-package*)*, 3
extraChIPs-package, 3

facet_grid, *68*
filterWindowsControl, *21, 22*
filterWindowsProportion, *21, 22*
findOverlaps, *11*, *38*, *40*, *43*
fitAssayDiff, 24
fitAssayDiff(), *4*
fitAssayDiff,SummarizedExperiment-method *(*fitAssayDiff*)*, 24
fixed_width_datasets, 26

GeneRegionTrack, *55–57*
geom_boxplot, *50*
geom_density, *45*
geom_label, *66*
geom_line, *45*
geom_point, *48*
geom_smooth, *62*
geom_text, *48*, *73*
getBestTest, *43*
getProfileData, 27, *69*
getProfileData(), *4*
getProfileData,BigWigFile,GenomicRanges-method *(*getProfileData*)*, 27
getProfileData,BigWigFileList,GenomicRanges-method *(*getProfileData*)*, 27
getProfileData,character,GenomicRanges-method *(*getProfileData*)*, 27
GInteractions, *10*
glmQLFit, *25*
glmQLFTest, *26*
glmTreat, *25, 26*
glue, *31*, *65, 66*, *72, 73*
grch37.cytobands, *56*
grch37.cytobands *(*cytobands*)*, 16
grch38.cytobands, *56*
grch38.cytobands *(*cytobands*)*, 16
grlToSE, 29
grlToSE(), *3*
grlToSE,GRangesList-method *(*grlToSE*)*, 29

IdeogramTrack, *17*, *56*
importPeaks, 31
importPeaks(), *4*
InteractionTrack, *56*
intersect, *76*
intersectMC, *32, 33*
intersectMC *(*setoptsMC*)*, 76
intersectMC(), *3*
intersectMC,GRanges,GRanges-method *(*setoptsMC*)*, 76
IRanges::CompressedList, *3*

lfcShrink, *25, 26*
limma-trend, *25*
lmFit, *25*, *79*

makeConsensus, 32, *60*
makeConsensus(), *4*, *62*
mapByFeature, 33
mapByFeature(), *3*
mapGrlCols, 36
mergeByCol, 37
mergeByCol(), *3*
mergeByCol,GenomicRanges-method *(*mergeByCol*)*, 37
mergeByCol,RangedSummarizedExperiment-method *(*mergeByCol*)*, 37
mergeByHMP, 39
mergeByHMP,GenomicRanges-method *(*mergeByHMP*)*, 39
mergeByHMP,RangedSummarizedExperiment-method *(*mergeByHMP*)*, 39
mergeBySig, 41
mergeBySig,GenomicRanges-method *(*mergeBySig*)*, 41
mergeBySig,RangedSummarizedExperiment-method *(*mergeBySig*)*, 41
minimalTests, *43*
MulticoreParam, *29*

nbinomWaldTest, *25, 26*
normalizeToMatrix, *28*
normLibSizes, *25, 26*

overlapsAny, *75*

p.adjust.methods, *38*
p.hmp, *41*
partitionRanges, 43

partitionRanges(), *3*
partitionRanges,GRanges,GRanges-method
(partitionRanges), 43
peaks (fixed_width_datasets), 26
plot_layout, *73*
plotAssayDensities, 45
plotAssayDensities(), *4*
plotAssayDensities,SummarizedExperiment-method
(plotAssayDensities), 45
plotAssayHeatmap, 46
plotAssayHeatmap,SummarizedExperiment-method
(plotAssayHeatmap), 46
plotAssayPCA, 48
plotAssayPCA(), *4*
plotAssayPCA,SummarizedExperiment-method
(plotAssayPCA), 48
plotAssayRle, 49
plotAssayRle(), *4*
plotAssayRle,SummarizedExperiment-method
(plotAssayRle), 49
plotGrlCol, 51
plotHFGC, 53
plotHFGC(), *4*
plotOverlaps, 59
plotOverlaps(), *4*
plotOverlaps,GRangesList-method
(plotOverlaps), 59
plotOverlaps,list-method
(plotOverlaps), 59
plotPairwise, 61
plotPie, 64
plotPie(), *4*
plotPie,data.frame-method (plotPie), 64
plotPie,DataFrame-method (plotPie), 64
plotPie,GRanges-method (plotPie), 64
plotProfileHeatmap, 67
plotProfileHeatmap(), *4*
plotProfileHeatmap,GenomicRanges-method
(plotProfileHeatmap), 67
plotProfileHeatmap,GenomicRangesList-method
(plotProfileHeatmap), 67
plotSplitDonut, 70
plotSplitDonut(), *4*
plotSplitDonut,data.frame-method
(plotSplitDonut), 70
plotSplitDonut,DataFrame-method
(plotSplitDonut), 70
plotSplitDonut,GRanges-method
(plotSplitDonut), 70
plotTracks, *55*
prcomp, *49*
propOverlap, 74
propOverlap(), *3*
propOverlap,GRanges,GRanges-method
(propOverlap), 74

RangedSummarizedExperiment, *21, 22*
reduce, *30*, *32*, *38*, *40*, *43*, *60*, *75*
reduceMC, *32, 33*, 75
reduceMC(), *3*

se (fixed_width_datasets), 26
seqinfo, *31*
setdiff, *76*
setdiffMC (setoptsMC), 76
setdiffMC(), *3*
setdiffMC,GRanges,GRanges-method
(setoptsMC), 76
setoptsMC, 76
settings, *57*
simpleUpSet, *60*
stitchRanges, 77
stitchRanges(), *3*
str_sort, *15*
SummarizedExperiment::RangedSummarizedExperiment,
3, 4

tibble, *10*
tibble::as_tibble(), *9*
tibble::tibble, *3*
treat, *25, 26*

union, *76*
unionMC (setoptsMC), 76
unionMC(), *3*
unionMC,GRanges,GRanges-method
(setoptsMC), 76

voom, *79*
voomWeightsFromCPM, 78
voomWithQualityWeights, *79*