# Package ‘extraChIPs’

October 18, 2022

**Version** 1.0.4

**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 with 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/steveped/extraChIPs`

**BugReports** `https://github.com/steveped/extraChIPs/issues`

**Depends** BiocParallel, R (>= 4.2.0), GenomicRanges, ggplot2, SummarizedExperiment, tibble

**Imports** BiocIO, broom, ComplexUpset, csaw, dplyr, edgeR, EnrichedHeatmap, forcats, GenomeInfoDb, GenomicInteractions, ggforce, ggrepel, ggside, glue, grDevices, grid, Gviz, InteractionSet, IRanges, limma, methods, RColorBrewer, rlang, Rsamtools, rtracklayer, S4Vectors, scales, stats, stringr, tidyr, tidyselect, utils, vctrs, VennDiagram

**Suggests** BiocStyle, covr, knitr, plyranges, rmarkdown, testthat (>= 3.0.0), tidyverse

**biocViews** ChIPSeq, HiC, Sequencing, Coverage

**BiocType** Software

**VignetteBuilder** knitr

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.1

**Config/testthat/edition** 3

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

**git_branch** RELEASE_3_15

**git_last_commit** 303de4a

**git_last_commit_date** 2022-08-30

**Date/Publication** 2022-10-18

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

**Maintainer** Stephen Pederson <`stephen.pederson.au@gmail.com`>

## R topics documented:

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.
- `plotProfileHeatmap()` plots the average signal around a set of ranges, as prepared by `getProfileData()`
- `plotPie()` emables simple comparison across multuple annotation columns within a GRanges object.
- `plotAssayDensities()`, `plotAssayPCA()` and `plotAssayRle()` provide simple interfaces to plotting key values from a SummarizedExperiment::RangedSummarizedExperiment.

### Convenience Functions

- `collapseGenes()` prints a vector of genes for an rmarkdown document, using italics.
- `importPeaks()` imports large numbers of broadPeak or narrowPeak files
- `voomWeightsFromCPM()` allows creation of an limma::EList object as would be created from counts by `limma::voom()`, but using `edgeR::cpm()` values as input.

### Author(s)

Stephen Pederson

---

## `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"), ...)
```

### 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

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)
```

---

## 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, ...)

## 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 GRanges object |

### 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)
```

## `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,
  ref,
  q = 0.5,
  logCPM = TRUE,
  keep.totals = TRUE,
  BPPARAM = bpparam()
)
```

### Arguments

| | |
|---|---|
| `x` | RangedSummarizedExperiment containing sample counts |
| `bg` | RangedSummarizedExperiment containing background/input counts |
| `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 |
| `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.

**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 conde.

## 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[, !is.na(wincounts$treat)],
    bg = wincounts[, is.na(wincounts$treat)],
    ref = peaks,
    q = 0.8 # Better to use q = 0.5 on real data
)
```

## ex_datasets *Datasets for an example region*

### Description

Various example datasets for demonstrating 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
```

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,
  ...
)

## 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 |
| `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 extensions 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 RangesSummarizedEperiment

### 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 or broadPeak format

## Usage

```
importPeaks(
  x,
  type = c("narrow", "broad"),
  blacklist,
  seqinfo,
  pruning.mode = c("coarse", "error"),
  sort = TRUE,
  setNames = 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 |
| `...` | passed to `sort` |

## Details

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

## Value

A GRangesList

## Examples

```
fl <- system.file(
"extdata", "testFiles", "test.narrowPeak", package = "extraChIPs"
)
gr <- importPeaks(fl, "narrow")
```

---

`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 nad 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)
```

---

`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 = FALSE,
  ...
)

## 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 |

### 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"))
df <- DataFrame(logFC = rnorm(3), logCPM = rnorm(3,8), p = 10^-rexp(3))
mergeByCol(x, df, col = "logCPM", pval = "p")
mcols(x) <- df
x
mergeByCol(x, col = "logCPM", pval = "p")
```

---

`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",
  colour = NULL,
  linetype = NULL,
  trans = NULL,
  n_max = Inf,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | A SummarizedExperiment object |
| `...` | Not used |
| `assay` | An assay within x |
| `colour` | The column in colData to colour lines by |
| `linetype` | Any column in colData used to determine linetype |
| `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. (See example)

## Examples

```
nrows <- 200; ncols <- 4
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
df <- DataFrame(treat = c("A", "A", "B", "B"))
se <- SummarizedExperiment(
  assays = SimpleList(counts = counts),
  colData = df
)
plotAssayDensities(se, colour = "treat") +
  labs(colour = "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 = c(),
  shape = c(),
  label = c(),
  show_points = TRUE,
  pc_x = 1,
  pc_y = 2,
  trans = c(),
  n_max = Inf,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | An object containing an assay slot |
| `...` | Not used |
| `assay` | The assay to perform PCA on |
| `colour` | The column name to be used for colours |
| `shape` | The column name to be used for determining the shape of points |
| `label` | The column name to be used for labels |
| `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 |

## 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

```
nrows <- 200; ncols <- 4
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
df <- DataFrame(treat = c("A", "A", "B", "B"), sample = seq_len(4))
se <- SummarizedExperiment(
  assays = SimpleList(counts = counts),
  colData = df
)
plotAssayPCA(se, "counts", colour = "treat", label = "sample")
```

---

`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 = c(),
  fill = c(),
  rle_group = c(),
  x_col = "sample",
  n_max = Inf,
  trans = c(),
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | A SummarizedExperiment object |
| `...` | Not used |
| `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 |
| `x_col` | Any alternative columns in `colData(x)` to use for the x-axis. Commonly an alternative sample label. |

| | |
|---|---|
| `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

```
nrows <- 200; ncols <- 4
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
df <- DataFrame(treat = c("A", "A", "B", "B"))
se <- SummarizedExperiment(
  assays = SimpleList(counts = counts),
  colData = df
)
plotAssayRle(se, "counts", fill = "treat")
```

---

# `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 = 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 |
| `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 |
| `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

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"),
  set_col = NULL,
  ...,
  .sort_sets = "ascending",
  min.gapwidth = 1L,
  ignore.strand = TRUE
)

## S4 method for signature 'list'
plotOverlaps(
  x,
  type = c("auto", "venn", "upset"),
  set_col = NULL,
  ...,
  .sort_sets = "ascending"
)
```

## 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 upset 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 |
| `set_col` | Colours to be assigned to each set |
| `.sort_sets` | passed to `sort_sets` in upset |
| `min.gapwidth, ignore.strand` | Passed to reduce |

## 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, type = "upset", set_col = 1:3, labeller = stringr::str_to_title)
plotOverlaps(ex[1:2])

## GRangesList object will produce a boxplot of summarised values in the
## upper panel
set.seed(100)
grl <- GRangesList(
  a = GRanges(c("chr1:1-10", "chr1:21-30", "chr1:31-40")),
  b = GRanges(c("chr1:12-15", "chr1:21-30", "chr1:46-50"))
)
grl$a$score <- rnorm(3)
grl$b$score <- rnorm(3)
plotOverlaps(grl, type = 'upset', var = 'score')
```

---

`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,
  width = 0.8,
  show_total = TRUE,
  label_fill = "white",
  label_alpha = 1,
  label_size = 3,
  min_p = 0.01,
  show_category = TRUE,
  category_size = 3,
  category_colour = "black",
  category_width = 15,
  ...
)
```

## 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 |
| `width` | Scale the width of all pies |
| `show_total` | logical(1) Show labels on each pie chart with the tally for that complete chart |
| `label_fill` | The background colour for tally labels |
| `label_alpha` | Transparency for tally labels |
| `label_size` | Size of the tally labels. Passed to geom_label |
| `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 |
| `show_category` | Show category labels around the edge of the plot if only one category/column is specified |
| `category_size` | The size of category labels if only one category/column is specified |
| `category_colour` | The colour of category labels if only one column is specified |
| `category_width` | Width at which category labels will wrap onto a new line |

## 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")
plotPie(df, fill = "feature", scale_by = "w")
plotPie(df, fill = "feature", x = "TF1")
plotPie(df, fill = "feature", x = "TF1", y = "TF2") +
 scale_fill_viridis_d() +
 theme_bw()

## Manually adding percentages
plotPie(df, fill = "feature", x = "TF1", label_size = 5) +
  geom_label(
   aes(x_lab, y_lab, label = lab),
   data = . %>%
     dplyr::mutate(
       x_lab = x + 0.5*r*sin(label_radians),
       y_lab = 1 + 0.5*r*cos(label_radians),
       lab = scales::percent(p, 0.1)
     ),
   size = 3.5
 )

## 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")
## Show values scaled by width of each range
plotPie(gr, fill = "feature", scale_by = "width")
```

## `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,
  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,
  relHeight = 0.3,
  ...
)

## S4 method for signature 'GenomicRanges'
plotProfileHeatmap(
  object,
  profileCol,
  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,
  relHeight = 0.3,
  ...
)
```

## 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` | Column used for facetting 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 |
| `relHeight` | The relative height of the top summary panel. Represents the fraction of the plotting area taken up by the summary panel. |

## 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")
```

---

## 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 the `voom` function. 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

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

# Index