Package: swissknife (via r-universe)

September 16, 2024

Title Handy code shared in the FMI CompBio group

Version 0.41

Description A collection of useful R functions performing various tasks that might be re-usable and worth sharing.

SystemRequirements C++11

Depends R (>= 3.5.0)

Imports BiocGenerics, BiocParallel, datasets, GenomeInfoDb, GenomicRanges, graphics, grDevices, IRanges, KernSmooth, matrixStats, methods, png, Rcpp, S4Vectors, stats, usethis, SummarizedExperiment, utils, XVector, dplyr

Suggests Biostrings, BiocStyle, BSgenome, GenomicAlignments, Gviz, knitr, QuasR, Rbowtie, rlang, rmarkdown, Rsamtools, rtracklayer, SingleCellExperiment, SingleR, testthat, tidyr, withr, wordspace, BiocManager

LinkingTo Rcpp

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

VignetteBuilder knitr

URL https://github.com/fmicompbio/swissknife

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

RemoteUrl https://github.com/fmicompbio/swissknife

RemoteRef HEAD

RemoteSha bbfe1f9020d9a1ae2a582bbf3759a2b46499a469

Contents

`swissknife-package` *swissknife - handy code shared in the FMI CompBio group*

Description

swissknife is a collection of useful R functions performing various tasks that might be re-usable and worth sharing.

Author(s)

Maintainer: Michael Stadler <michael.stadler@fmi.ch>

Authors:

- Charlotte Soneson <charlotte.soneson@fmi.ch>
- Panagiotis Papasaikas <panagiotis.papasaikas@fmi.ch>
- Dania Machlab <dania.machlab@fmi.ch>
- Fiona Ross <fiona.ross@fmi.ch>

Other contributors:

- Friedrich Miescher Institute for Biomedical Research [copyright holder]

See Also

Useful links:

- https://github.com/fmicompbio/swissknife

`addUtilsFunctions` *Copy utility functions to package*

Description

This function copies handy utility functions to a new script in a specified location. Currently, the script contains the following utility functions:

- `.assertScalar()` - convenience function to check the validity of scalar variables.
- `.assertVector()` - convenience function to check the validity of vector variables.

Usage

```
addUtilsFunctions(outFile = "R/utils.R", copyTests = TRUE)
```

Arguments

`outFile`	Character scalar, giving the path to which the script should be copied. The path is relative to the root of the active project. If a file with this name already exists, the function will ask for confirmation before overwriting it.
`copyTests`	Logical scalar, defining whether to copy unit tests for the utility functions to `tests/testthat/test-<name>.R`, where `<name>` is the base name of `outFile`. If the target package is not yet set up to use `testthat`, the function will also run `usethis::use_testthat()` to generate the required folder structure and add `testthat` to the list of suggested package in the DESCRIPTION file.

Author(s)

Charlotte Soneson

annotateRegions *Annotate regions.*

Description

Annotate a GRanges object with sets of reference GRanges or GRangesList objects, with respect to overlaps and nearest neighbors.

Usage

```
annotateRegions(
  x,
  hasOverlap = list(),
  fracOverlap = list(),
  numOverlap = list(),
  nearest = list(),
  ignore.strand = TRUE
)
```

Arguments

x	The GRanges object to annotate.
hasOverlap	Named list with GRanges or GRangesList object(s). For each list element, a logical vector "X.hasOverlap" will be added to the mcols of the result, with TRUE for each tile that overlaps any region in that element. "X" is obtained from names(hasOverlap).
fracOverlap	Named list with GRanges or GRangesList object(s). For each list element, a numeric vector "X.fracOverlap" will be added to the mcols of the result, with a value between 0 and 1 giving the fraction of bases in a tile that overlaps with any region in that element. "X" is obtained from names(fracOverlap).
numOverlap	Named list with GRanges or GRangesList object(s). For each list element, two numeric vectors "X.numOverlapWithin" and "X.numOverlapAny" will be added to the mcols of the result, giving the number of ranges in that element that are fully contained within a tile, or that overlap with a tile in any way, respectively. "X" is obtained from names(numOverlap).
nearest	Named list with GRanges or GRangesList object(s). For each list element, two numeric vectors "X.nearestName" and "X.nearestDistance" will be added to the mcols of the result, giving the name and distance of the nearest range in that element for each tile. "X" is obtained from names(nearest), and the values of "X.nearestName" from names(nearest$X). If multiple nearest ranges are at the same distance from a tile, an arbitrary one is reported in "X.nearestName".
ignore.strand	Logical scalar passed to findOverlaps when searching for overlaps between x and reference regions.

Value

A GRanges similar to x, with annotations added to its metadata columns (mcols).

Author(s)

Michael Stadler

See Also

`getGenomicTiles` that uses this function, `findOverlaps` and `nearest` in package **GenomicRanges** used internally.

Examples

```
library(GenomicRanges)

x <- GRanges("chr1", IRanges(c(1, 12), width = 10))
tss <- GRanges("chr1", IRanges(c(1, 10, 30), width = 1,
                               names = paste0("t", 1:3)))
blacklist <- GRanges("chr1", IRanges(20, width = 5))
annotateRegions(x, hasOverlap = list(Blacklist = blacklist),
                fracOverlap = list(Blacklist = blacklist),
                numOverlap = list(TSS = tss),
                nearest = list(TSS = tss))
```

---

calcAndCountDist *Count frequency of differences between values in integer vectors.*

Description

Given two ascendingly sorted integer vectors `query` and `reference`, calculate and count the differences between their elements that are greater than zero and less than `maxd`. The number of observed distances `d` are reported in `cnt[d]`, and `maxd` corresponds to the `length(cnt)`. The function is called by `calcPhasogram`, which provides a higher level, more conventient interface.

Usage

```
calcAndCountDist(query, reference, cnt)
```

Arguments

`query`	first `integer` vector.
`reference`	second `integer` vector. Distances are calculated from each element in `query` to each greater element in `reference`.
`cnt`	`NumericVector` to store the result in. The length of `cnt` defines the maximal distance that will be included in the analysis, and new counts will be added to the values of `cnt`.

Value

`numeric` vector `cnt`, where `cnt[d]` correspond to the number of observed distances `d`.

Author(s)

Michael Stadler

`calcPhasogram` *Calculate phasograms (same strand alignment distances).*

Description

Calculate the frequencies of same strand alignment distances, for example from MNase-seq data to estimate nucleosome repeat length. Distance calculations are implemented in C++ (`calcAndCountDist`) for efficiency.

Usage

```
calcPhasogram(fname, regions = NULL, rmdup = TRUE, dmax = 3000L)
```

Arguments

`fname`	`character` vector with one or several bam files. If multiple files are given, distance counts from all will be summed.
`regions`	`GRanges` object. Only alignments falling into these regions will be used. If `NULL` (the default), all alignments are used.
`rmdup`	`logical(1)` indicating if duplicates should be removed. If `TRUE` (the default), only one of several alignments starting at the same coordinate is used.
`dmax`	`numeric(1)` specifying the maximal distance between same strand alignments to count.

Value

`integer` vector with `dmax` elements, with the element at position `d` giving the observed number of alignment pairs at that distance.

Author(s)

Michael Stadler

References

Phasograms were originally described in Valouev et al., Nature 2011 (doi:10.1038/nature10002). The implementation here differs in two ways from the original algorithms:

1. It does not implement removing of positions that have been seen less than `n` times (referred to as a `n`-pile subset in the paper).
2. It does allow to retain only alignments that fall into selected genomic intervals (`regions` argument).

See Also

estimateNRL to estimate the nucleosome repeat length from a phasogram, plotPhasogram to visualize an annotated phasogram, calcAndCountDist for low-level distance counting.

Examples

```
if (requireNamespace("GenomicAlignments", quietly = TRUE) &&
    requireNamespace("Rsamtools", quietly = TRUE)) {
    bamf <- system.file("extdata", "phasograms", "mnase_mm10.bam",
                        package = "swissknife")
    pg <- calcPhasogram(bamf)
    print(estimateNRL(pg, usePeaks = 1:4)[1:2])
    plotPhasogram(pg, usePeaks = 1:4, xlim = c(0,1000))
}
```

---

col2hex *Get hex color*

Description

The function returns a color in hex form given a valid name of a color in R.

Usage

```
col2hex(col, alpha = 255)
```

Arguments

col	a character, integer or vector of both types containing the names of the colors or colors as integers.
alpha	a numerical value in the range [0,1] or [0,255] that indicates the transparency of the color(s). If the given values are between 0 and 1, they are mapped to be between 0 and 255. An alpha value of 1 assumes the [0,1] range and provides maximum color. The default is set to 255.

Value

a character or character vector with the hex colors.

Author(s)

Dania Machlab

Examples

```
y <- rnorm(1000,0,1)
cols <- rep("red", length(y))
alpha <- seq(0,1,length.out=length(y))
hexcols <- col2hex(cols, alpha)
plot(1:length(y), y, bg=hexcols, pch=21)

y <- rnorm(1000,0,1)
cols <- rep("red", length(y))
alpha <- seq(0,255,length.out=length(y))
hexcols <- col2hex(cols, alpha)
plot(1:length(y), y, bg=hexcols, pch=21)
```

`estimateNRL` *Estimate the nucleosome repeat length (NRL) from a phasogram.*

Description

Estimate the nucleosome repeat length (NRL) from the frequencies of same-strand alignment distances (phasogram), e.g. generated by `calcPhasogram`. The NRL is obtained from the slope of a linear fit to the modes in the phasogram.

Usage

```
estimateNRL(
  x,
  mind = 140L,
  usePeaks = 1:8,
  span1 = 100/length(x),
  span2 = 1500/length(x)
)
```

Arguments

`x`	`numeric` vector giving the counts of alignment distances (typically the output of `calcPhasogram`.
`mind`	`integer(1)` specifying the minimal distance to be used for NRL estimation. The default value (140) ignores any distance too short to span at least a single nucleosome.
`usePeaks`	`integer` vector selecting the modes (peaks) in the phasogram used in NRL estimation.
`span1`	`numeric(1)` giving the smoothing parameter for de-trending loess fit (high pass filter).
`span2`	`numeric(1)` giving the smoothing parameter for de-noising loess fit (low pass filter).

Value

A `list` with elements:

nrl the estimated nucleosome repeat length

nrl.CI95 the 95% confidence interval

xs smoothed (de-trended) phasogram

loessfit the de-noising fit to the de-trended phasogram

lmfit the linear fit to the phasogram peaks

peaks the peak locations

mind minimal distance included in the fit

span1 smoothing parameter for de-trending loess fit

span2 smoothing parameter for de-noising loess fit

usePeaks the peaks used in the fit

Author(s)

Michael Stadler

See Also

`calcPhasogram` to calculate the phasogram from alignments, `plotPhasogram` to visualize an annotated phasogram

Examples

```
# see the help for calcPhasogram() for a full example
```

`getGenomicTiles` *Get regions tiling a genome.*

Description

Get sequential, potentially annotated regions of a fixed lengths (tiles) along chromosomes of a genome.

Usage

```
getGenomicTiles(
  genome,
  tileWidth,
  hasOverlap = list(),
  fracOverlap = list(),
  numOverlap = list(),
  nearest = list(),
  addSeqComp = TRUE
)
```

Arguments

genome	The genome to work on. Either a BSgenome object, a character scalar with the name of an installed BSgenome or with a file path and name pointing to a fasta file with the genome sequence, or a named numeric vector giving the names and lengths of chromosomes.
tileWidth	numeric scalar with the tile length.
hasOverlap, fracOverlap, numOverlap, nearest	Named lists with GRanges or GRangesList object(s) used to annotate genomic tiles. See annotateRegions for details.
addSeqComp	logical scalar. If TRUE and primary sequence can be obtained from genome, also add sequence composition features for each tile to the annotations. Currently, the following features are included: percent of G+C bases ("percGC"), CpG observed-over-expected ratio ("CpGoe").

Details

The last tile in each chromosome is dropped if it would be shorter than tileWidth. Generated tiles are unstranded (*) and therefore overlaps or searching for nearest neighbors are ignoring strands of annotations (ignore.strand=TRUE).

Value

A GRanges object with genome tiling regions. Optional tile annotations are contained in its metadata columns (mcols).

Author(s)

Michael Stadler

See Also

tileGenome and annotateRegions used by getGenomicTiles internally.

Examples

```
library(GenomicRanges)

tss <- GRanges("chr1", IRanges(c(1, 10, 30), width = 1,
                               names = paste0("t", 1:3)))
blacklist <- GRanges("chr1", IRanges(20, width = 5))
getGenomicTiles(c(chr1 = 45, chr2 = 12), tileWidth = 10,
                hasOverlap = list(Blacklist = blacklist),
                fracOverlap = list(Blacklist = blacklist),
                numOverlap = list(TSS = tss),
                nearest = list(TSS = tss))
```

getInsertSizeDistFromBam *Tabulate insert sizes from paired-end alignments in bam files.*

Description

Read and tabulate the insert sizes from paired-end alignments contained in one or several bam files. By default, all properly aligned read pairs are included. Optionally, alignments can be restricted to those in a specific genomic region (`regions` argument) or the number of alignments read can be limited (`nmax` argument).

Usage

```
getInsertSizeDistFromBam(
  fname,
  regions = NULL,
  nmax = NA_integer_,
  isizemax = 800,
  exclude = c("chrM", "chrY", "chrX")
)
```

Arguments

`fname`	`character` vector with paths to one or several bam files. If multiple files are given, insert sizes from all will be pooled and tabulated together.
`regions`	`GRanges` object. Only alignments falling into these regions will be used. If `NULL` (the default), all alignments are used.
`nmax`	`numeric(1)` specifying the maximal number of alignments to read. If `NA` (the default), the alignments in `regions` (if `regions` are not `NULL`) or in the bam file will be used.
`isizemax`	`numeric(1)` specifying the maximal insert size to report. Larger insert sizes will be set to `isizemax` with on their number will be reported.
`exclude`	`character` vector with chromosome names to be excluded. Alignments on these chromosomes will be excluded. `exclude` will be ignored if `regions` is not `NULL`.

Value

`integer` vector with the number of insert sizes. The element at position `i` gives the observed number of alignment pairs with an insert size of `i`. The number of insert sizes greater than `isizemax` that were set to `isizemax` are reported in the attribute `"ncapped"`.

Author(s)

Michael Stadler

See Also

scanBam used to read alignments.

Examples

```
if (requireNamespace("Rsamtools", quietly = TRUE)) {
    bamf <- system.file("extdata", "getInsertSizeDistFromBam", "atac_mm10.bam",
                        package = "swissknife")
    isize <- getInsertSizeDistFromBam(bamf)
    attr(isize, "ncapped")
    plot(isize, type = "l",
         xlab = "Insert size (bp)", ylab = "Number of fragments")
}
```

---

getMappableRegions *Get mappable regions of a genome.*

Description

Given a k-mer length and the maximum number of allowed hits per k-mer, find all mappable regions in a genome.

Usage

```
getMappableRegions(
  genome,
  genomeIndex,
  kmerLength = 50,
  maxHits = 1,
  Ncpu = 2,
  quiet = TRUE
)
```

Arguments

genome	The genome sequence to work on. Either a BSgenome object, a character scalar with the name of an installed BSgenome or with a file path and name pointing to a fasta file with the genome sequence.
genomeIndex	character scalar with the path to the bowtie index and prefix to align against, in the form </path/to/index>/<prefix>, or the name of an installed Rbowtie index package created by the **QuasR** package for an installed BSgenome package.
kmerLength	numeric scalar specifying the k-mer length (width of overlapping windows in genome), usually set to the typical read length for which to get the mappable regions.

maxHits	numeric scalar specifying the maximum number of hits (matches) of a k-mer in the genome to be considered mappable.
Ncpu	numeric scalar specifying the number of CPU threads to use for alignments.
quiet	logical scalar indicating if progress information should be printed on the console.

Details

Sequences of all overlapping windows are extracted from the genome and aligned to the provided genome index using bowtie with parameters -f -v 0 -a -B 1 -m maxHits. If no more than maxHits hits are found, the window is defined mappable.

Value

A GRanges object with mappable regions. All plus-strand sequences in genome of length kmerLength with their start (leftmost) position overlapping the GRanges object do not generate more than maxHits hits when aligned to the genome.

Author(s)

Michael Stadler

See Also

bowtie in package **Rbowtie** used by getMappableRegions to align reads to the genome; bowtie_build in package **Rbowtie** for indexing a genome.

Examples

```
if (requireNamespace("Rbowtie", quietly = TRUE)) {
    library(Rbowtie)

   genomefile <- system.file("extdata", "getMappableRegions", "hg19sub.fa", package = "swissknife")
    indexdir <- tempfile()
    indexpre <- "index"
    indexname <- file.path(indexdir, indexpre)
    idx <- bowtie_build(genomefile, indexdir)

    mapgr <- getMappableRegions(genomefile, indexname, 50, quiet = FALSE)
    print(mapgr)
}
```

`labelCells` *Assign labels to cells using known marker genes*

Description

Given marker gene sets for cell types, identify cells with high expression of the marker genes (positive examples), then use these cells to create a reference transcriptome profile for each cell type and identify additional cells of each type using `SingleR`. These marker genes should specifically expressed a single cell type, e.g. CD3 which is expressed by all T cell subtypes would not be suitable for specific T cell subtypes.

Usage

```
labelCells(
  sce,
  markergenes,
  fraction_topscoring = 0.01,
  expr_values = "logcounts",
  normGenesetExpressionParams = list(R = 200),
  aggregateReferenceParams = list(power = 0.5),
  SingleRParams = list(),
  BPPARAM = SerialParam()
)
```

Arguments

`sce`	`SingleCellExperiment` object.
`markergenes`	Named `list` of `character` vectors with the marker genes for each cell types. The marker genes must be a subset of `rownames(sce)`.
`fraction_topscoring`	`numeric` vector of length 1 or the same length as `markergenes` giving the fraction(s) of top scoring cells for each cell type to pick to create the reference transcriptome profile.
`expr_values`	Integer scalar or string indicating which assay of `sce` contains the expression values.
`normGenesetExpressionParams`	`list` with additional parameters for `normGenesetExpression`.
`aggregateReferenceParams`	`list` with additional parameters for `aggregateReference`.
`SingleRParams`	`list` with additional parameters for `SingleR`.
`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation.

Value

A `list` of three elements named `cells`, `refs` and `labels`. `cells` contains a `list` with the numerical indices of the top scoring cells for each cell type. `refs` contains the pseudo-bulk transcriptome profiles used as a reference for label assignment, as returned by `aggregateReference`. `labels` contains a `DataFrame` with the annotation statistics for each cell (one cell per row), generated by `SingleR`.

Author(s)

Michael Stadler

See Also

`normGenesetExpression` used to calculate scores for marker gene sets; `aggregateReference` used to create reference profiles; `SingleR` used to assign labels to cells.

Examples

```
if (requireNamespace("SingleR", quietly = TRUE) &&
    requireNamespace("SingleCellExperiment", quietly = TRUE)) {

    # create SingleCellExperiment with cell-type specific genes
    library(SingleCellExperiment)
    n_types <- 3
    n_per_type <- 30
    n_cells <- n_types * n_per_type
    n_genes <- 500
    fraction_specific <- 0.1
    n_specific <- round(n_genes * fraction_specific)

    set.seed(42)
    mu <- ceiling(runif(n = n_genes, min = 0, max = 30))
    u <- do.call(rbind, lapply(mu, function(x) rpois(n_cells, lambda = x)))
    rownames(u) <- paste0("g", seq.int(nrow(u)))
    celltype.labels <- rep(paste0("t", seq.int(n_types)), each = n_per_type)
    celltype.genes <- split(sample(rownames(u), size = n_types * n_specific),
                            rep(paste0("t", seq.int(n_types)), each = n_specific))
    for (i in seq_along(celltype.genes)) {
        j <- celltype.genes[[i]]
        k <- celltype.labels == paste0("t", i)
        u[j, k] <- 2 * u[j, k]
    }
    v <- log2(u + 1)
    sce <- SingleCellExperiment(assays=list(counts=u, logcounts=v))

    # define marker genes (subset of true cell-type-specific genes)
    marker.genes <- lapply(celltype.genes, "[", 1:5)
    marker.genes

    # predict cell types
    res <- labelCells(sce, marker.genes,
```

```
                    fraction_topscoring = 0.1,
                    normGenesetExpressionParams = list(R = 50))

    # high-scoring cells used as references for each celltype
    res$cells

    # ... from these, pseudo-bulks were created:
    res$refs

    # ... and used to predict labels for all cells
    res$labels$pruned.labels

    # compare predicted to true cell types
    table(true = celltype.labels, predicted = res$labels$pruned.labels)
}
```

`loadExampleData` *Access example data*

Description

Make example data available, typically for use in teaching.

Usage

```
loadExampleData(name = "list", envir = globalenv(), verbose = TRUE)
```

Arguments

`name`	An optional character scalar specifying the data set(s) to be made available. The special name `"list"` (default) is used to print a data frame of available data sets with descriptions. The special name `"latest"` will select the latest data set(s) available.
`envir`	specifies the environment in which the data should be made available. By default, `envir = globalenv()`, which creates the example data objects in the user workspace. Possible alternative environment are for example `parent.frame()`, which is the environment in which `loadExampleData()` was called.
`verbose`	A logical scalar. If `TRUE`, report what is being selected and made available.

Value

A `data.frame` (invisibly) with one row for each dataset that was made available in the global environment.

Author(s)

Michael Stadler

Examples

```
loadExampleData()
loadExampleData("mycars")
```

`normGenesetExpression` *Calculate normalized expression of a gene set*

Description

Calculate normalized expression for a set of genes in each cell from a `SingleCellExperiment`, using random sets of similarly expressed genes as background to account for cell quality and sequencing depth.

Usage

```
normGenesetExpression(
  sce,
  genes,
  expr_values = "logcounts",
  subset.row = NULL,
  R = 200,
  BPPARAM = SerialParam()
)
```

Arguments

`sce`	`SingleCellExperiment` object.
`genes`	`character` vector with the genes in the set. Must be a subset of `rownames(sce)`.
`expr_values`	Integer scalar or string indicating which assay of `sce` contains the expression values.
`subset.row`	Sample random genes only from these. If `NULL` (the default), the function will sample from all genes in `sce`. Alternatively, `subset.row` can be a logical, integer or character vector indicating the rows (genes) of `sce` to use for sampling. This allows for example to exclude highly variable genes from the sampling which are likely expressed only in certain cell types.
`R`	Integer scalar giving the number of random gene sets to sample for normalization.
`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation.

Value

A `numeric` vector with normalized gene set scores for each cell in `sce`.

Author(s)

Michael Stadler

Examples

```
if (require(SingleCellExperiment)) {
    # get sce
    example(SingleCellExperiment, echo=FALSE)
    rownames(sce) <- paste0("g", seq.int(nrow(sce)))

    # calculate gene set expression scores
    markers <- c("g1", "g13", "g27")
    scores <- normGenesetExpression(sce, markers, R = 50)

    # compare expression of marker genes with scores
    plotdat <- cbind(scores, t(logcounts(sce)[markers, ]))
    cor(plotdat)
    pairs(plotdat)
}
```

`parsePkgVersions` *Parse R and R package versions from session informations*

Description

The function parses the R version and R package versions from session information (created by `sessionInfo()`, tested with R 3.6) in files provided in `infiles`. Two types of files are currently supported:

- Rout: Files containing R console output (created by `R CMD BATCH script.R output.Rout`
- md: Files containing markdown output created by `rmarkdown::render('input.Rmd', clean = FALSE)`, which will keep the intermediate `.md` file.

Usage

```
parsePkgVersions(infiles)
```

Arguments

`infiles`	Character vector with text files (extension must be either `.Rout` or `.md`), containing session information to parse out.

Value

A `list` of `lists` with one element in the outer list for each R version, contianing an innter list with elements `files` and `packages`.

Author(s)

Michael Stadler

Examples

```
f <- list.files(system.file("extdata", "parsePkgVersions",
                            package = "swissknife"),
                full.names = TRUE)
parsePkgVersions(f)
```

---

plotBitScatter *Create a bitmap-rendered plot.*

Description

`plotBitScatter` is a wrapper around `plot` which renders the plot area as a bitmap (png), but keeps all other elements (axes, labels, etc.) as vector elements. This is especially useful for keeping the size of PDF files with scatter plots with many elements small, while retaining editability of axes.

Usage

```
plotBitScatter(
  x,
  y = NULL,
  ...,
  densCols = TRUE,
 colpal = c("#00007F", "blue", "#007FFF", "cyan", "#7FFF7F", "yellow", "#FF7F00", "red",
    "#7F0000"),
  xpixels = 1000,
  ypixels = NULL,
  pointsize = NULL
)
```

Arguments

`x`	`numeric` vector with x-coordinates of points, or a two-column matrix with x- and y- coordinates.
`y`	`numeric` vector with y-coordinates of points (same length as `x`). Can be `NULL`, in which case `x` must be a two-column matrix.
`...`	any further arguments to be passed to `plot`
`densCols`	`logical(1)`. If `TRUE` and `col` is not given as an additional argument, then the local density of points will be used as colors, using the palette spanned by the colors in `colpal`.
`colpal`	vector of colors defining the palette for automatic density-based coloring.
`xpixels`	the number of pixels in the x dimension used for rendering the plotting area.

ypixels — the number of pixels in the y dimension used for rendering the plotting area. If NULL (the default), will be calculated automatically as xpixels * par('pin')[2] / par('pin')[1], such that the aspect ratio of the current plotting region is observed. This may not work (e.g. when using layout()), as this may result in negative values returned by par('pin'). In that case, ypixels should be set manually using this argument.

pointsize — the size of points used for the png device when rendering the plot. If NULL (the default), will be calculated automatically as 12 / graphics::grconvertX(par("pin")[1], from = "inches", to = "device") * xpixels. This may not work (e.g. when using layout()), as this may result in negative values returned by par('pin'). In that case, pointsize should be set manually using this argument.

Details

xpixels controls the resolution of the rendered plotting area. In order to keep circular plotting symbols circlular (e.g. pch = 1), ypixels is automatically calculated using xpixels and the aspect ratio of the current plotting area. If the plotting device is rescaled after calling plotBitScatter, circular plotting symbols may become skewed.

Value

NULL (invisibly)

Author(s)

Michael Stadler

Examples

```
x <- rnorm(1000)
y <- rnorm(1000)
par(mfrow=c(1,2))
plotBitScatter(x, y, main = "bitmap")
plot(x, y, main = "default")
```

---

plotGeneRegion *Plot gene region*

Description

Visualize the gene model for a gene of interest, or for all genes in a provided region, and/or show one or more coverage tracks based on bigwig file(s).

Usage

```
plotGeneRegion(
  gtf = "",
  granges = NULL,
  chr = "",
  start = NA_real_,
  end = NA_real_,
  showgene = "",
  bigwigFiles = "",
  bigwigCond = "",
  geneTrackTitle = "Genes",
  transcriptIdColumn = "transcript_id",
  geneIdColumn = "gene_id",
  geneSymbolColumn = "gene_name",
  lowerPadding = 0.15,
  upperPadding = 0.05,
  colorByStrand = FALSE,
 featureColors = c(plusmain = "#0E14D0", minusmain = "#D0350E", plusother = "#9E9BEB",
    minusother = "#DA907E"),
  condColors = NULL,
  scaleDataTracks = FALSE,
  plotTitle = NULL,
  ...
)
```

Arguments

gtf	Character scalar, path to gtf file (tested with Ensembl/Gencode files).
granges	GRanges object, typically generated from a GTF file using the prepareGTF function. This is an alternative to providing the link to the gtf file directly, and will take precedence over the gtf argument if provided.
chr	Character scalar, name of the chromosome to show.
start, end	Numeric scalars, start and end position of the region to show.
showgene	Character scalar, the gene ID/name to display. Will take precedence over positional range specification if provided.
bigwigFiles	Named character vector, paths to bigwig files.
bigwigCond	Named character vector, the grouping of the bigwig files (used for coloring of the coverage tracks).
geneTrackTitle	Character scalar, name of the gene track.
transcriptIdColumn	Character scalar, the column in the gtf file that contains the transcript ID. Passed to prepareGTF.
geneIdColumn	Character scalar, the column in the gtf file that contains the gene ID. Passed to prepareGTF.

`geneSymbolColumn`
: Character scalar, the column in the gtf file that contains the gene symbol (if available). Set to `""` if not available (in which case the gene IDs will be used in its place). Passed to `prepareGTF`.

`lowerPadding, upperPadding`
: Numeric scalars, setting the amount of padding in the lower and upper range of the plot, respectively. For example, a value of 0.05 will expand the range by 0.05 * (max coordinate - min coordinate) in the specified direction.

`colorByStrand`
: Logical scalar, determining whether gene features are colored by the annotated strand.

`featureColors`
: Named character vector of length 4, with elements `plusmain`, `minusmain`, `plusother`, `minusother`, giving the colors to use for the features if `colorByStrand` is TRUE.

`condColors`
: Either NULL or a named character vector (with the same names as the unique values of `bigwigCond`), giving the colors to use for the coverage tracks if `bigwigCond` is provided.

`scaleDataTracks`
: Logical scalar, indicating whether the data tracks should be scaled to have the same y-axis limits.

`plotTitle`
: Character scalar, the title of the final plot. If `NULL` (the default), it will be automatically defined based on the displayed gene or region.

`...`
: Additional arguments to be passed to `Gviz::plotTracks`.

Details

The gene annotation can be provided either as a path to a gtf file, or as a GRanges object (generated using the `prepareGTF` function to ensure compatibility). The region to display can be determined either by specifying a gene (ID or symbol) or by specifying a viewing range (chromosome, start and end positions).

Author(s)

Charlotte Soneson

Examples

```
if (requireNamespace("Gviz", quietly = TRUE)) {
    gtffile <- system.file("extdata/plotGeneRegion/mm10_ensembl98.gtf",
                           package = "swissknife")
    plotGeneRegion(gtf = gtffile,
                   showgene = "Tnfaip3")

    bwf <- system.file("extdata/plotGeneRegion/mnase_mm10.bw",
                       package = "swissknife")
    names(bwf) <- "bwf1"
    plotGeneRegion(gtf = gtffile,
                   bigwigFiles = bwf,
                   chr = "chr10", start = 20000000, end = 20005000)
    plotGeneRegion(bigwigFiles = bwf,
                   chr = "chr10", start = 20000000, end = 20005000)
```

```
    bwf2 <- c(bwf, bwf)
    names(bwf2) <- c("bwf1", "bwf2")
    bwc2 <- c("c1", "c2")
    names(bwc2) <- names(bwf2)
    plotGeneRegion(gtf = gtffile, bigwigFiles = bwf2, bigwigCond = bwc2,
                   showgene = "Map3k5")
}
```

`plotPhasogram` *Plot annotated phasogram.*

Description

Plot phasogram and annotate it with estimated nucleosome repeat length (NRL).

Usage

```
plotPhasogram(x, hide = TRUE, xlim = NULL, verbosePlot = FALSE, ...)
```

Arguments

`x`	`numeric` vector giving the counts of alignment distances (typically the output of `calcPhasogram`.
`hide`	If `TRUE` (the default), hide phasogram counts not used in the NRL estimate (`mind` parameter from `estimateNRL`).
`xlim`	`numeric(2)` with the x-axis (phase) limits in the first two plots (see Details). if `NULL` (the default), the full range defined by `x` and `hide` will be used.
`verbosePlot`	If `TRUE`, create three plots instead of just a single plot (see Details).
`...`	Additional arguments passed to `estimateNRL` to control NRL estimation.

Details

The function will visualize an annotated phasogram. For `verbosePlot=FALSE` (the default), it will create a single annotated plot. For `verbosePlot=TRUE`, it will create three plots (using `par(mfrow=c(1,3))`):

1. raw phase counts with de-trending and de-noising loess fits
2. residual phase counts with de-noising loess fit and detected peaks
3. linear fit to peaks and NRL estimation

Value

The return value from the call to `estimateNRL` (invisibly).

Author(s)

Michael Stadler

See Also

calcPhasogram to calculate the phasogram from alignments, estimateNRL to estimate nucleosome repeat length

Examples

```
# see the help for calcPhasogram() for a full example
```

---

plotSelVarGenes *Plot Selected Variable Genes*

Description

This function take the output from selVarGenes and plots the genes that have been selected to be highly variable across the cells. It plot the log2 coefficient of variation as a function of the log mean.

Usage

```
plotSelVarGenes(
  selVarGenes_list = NULL,
  xlab = "logMean",
  ylab = "logCV",
  main = "Selected Variable Genes",
  pch = 16,
  col = "#BEBEBE40",
  sel_col = "steelblue",
  colByBin = FALSE,
  asp = 1,
  ...
)
```

Arguments

selVarGenes_list	the output list from the selVarGenes function.
xlab	label for x-axis.
ylab	label for y-axis.
main	title for plot.
pch	point pch.
col	point color.
sel_col	point color of the selected variable genes.
colByBin	if TRUE, color the genes by the bin they've been assigned to.
asp	the y/x aspect ratio. Set to 1 when colByBin is TRUE.
...	additional parameters for the plot function.

Value

plot

Author(s)

Dania Machlab

Examples

```
if (requireNamespace("SingleCellExperiment", quietly = TRUE)) {
    # packages
    library(SingleCellExperiment)

    # create example count matrix
    # ... poisson distr per gene
    mu <- ceiling(runif(n = 2000, min = 0, max = 100))
    counts <- do.call(rbind, lapply(mu, function(x){rpois(1000, lambda = x)}))
    counts <- counts + 1
    # ... add signal to subset of genes (rows) and cells (columns)
    i <- sample(x = 1:nrow(counts), size = 500)
    j <- sample(x = 1:ncol(counts), size = 500)
    counts[i, j] <- counts[i, j] + sample(5:10, length(i), replace = TRUE)

    # create SCE
    sce <- SingleCellExperiment(list(counts = counts))

    # calculate sizeFactors
    libsizes <- colSums(counts)
    sizeFactors(sce) <- libsizes / mean(libsizes)

    # select variable genes
    varGenes <- selVarGenes(sce)

    # plot
    plotSelVarGenes(varGenes)
    plotSelVarGenes(varGenes, colByBin=TRUE)
}
```

---

prepareGTF *Prepare GTF file for use with plotGeneRegion*

Description

This function sets the names of the transcript and gene ID columns of the gtf file to "transcript" and "gene", removes version tags of the transcripts/genes and retains only the "exon" entries. The purpose is to make the file amenable to plotting with Gviz, using the `plotGeneRegion` function.

Usage

```
prepareGTF(
  gtf,
  transcriptIdColumn = "transcript_id",
  geneIdColumn = "gene_id",
  geneSymbolColumn = "gene_name"
)
```

Arguments

gtf	Character scalar, path to gtf file (tested with Ensembl/Gencode files).
transcriptIdColumn	Character scalar, the column in the gtf file that contains the transcript ID.
geneIdColumn	Character scalar, the column in the gtf file that contains the gene ID.
geneSymbolColumn	Character scalar, the column in the gtf file that contains the gene symbol (if available). Set to "" if not available (in which case the gene IDs will be used in its place).

Author(s)

Charlotte Soneson

Examples

```
gtf <- prepareGTF(gtf = system.file("extdata/plotGeneRegion/mm10_ensembl98.gtf",
                                    package = "swissknife"))
```

---

readSampleTsvs *Read sample tsv files from seqdata storage*

Description

The function searches the provided `seqdataDir` for tsv files corresponding to the provided `sampleIds` and returns a `data.frame` containing the metadata for all these samples.

Usage

```
readSampleTsvs(
  seqdataDir = "/tungstenfs/groups/gbioinfo/seqdata",
  sampleIds,
  keepMulti = TRUE,
  ...
)
```

Arguments

seqdataDir	Character scalar, the path to the directory containing the tsv files.
sampleIds	Character vector with sample IDs, which will be matched against the file names in seqDataDir. The sample IDs should not contain the .tsv suffix.
keepMulti	Logical scalar, indicating whether to keep samples that match more than one tsv file. If TRUE, these samples are represented by multiple rows in the table. If FALSE, these samples are excluded. In any case, a warning will be generated, listing the samples with multiple matching files.
...	Additional arguments that will be passed to list.files, e.g. to make the search case-insensitive or search recursively.

Value

A data.frame with metadata for the provided sampleIds.

Author(s)

Charlotte Soneson

Examples

```
if (requireNamespace("dplyr") && requireNamespace("tidyr")) {
    print(readSampleTsvs(seqdataDir = system.file("extdata/readSampleTsvs",
                                                  package = "swissknife"),
                         sampleIds = c("readSampleTsvsEx1",
                                       "readSampleTsvsEx2",
                                       "readSampleTsvsEx3")))
}
```

---

sampleControlElements *Sample control elements that match a target distribution.*

Description

Randomly sample from a set of control (background) elements, such that the selected elements are similarly distributed as a given set of target (foreground) elements.

Usage

```
sampleControlElements(
  x,
  idxTarget,
  idxControl = NULL,
  nbins = 50,
  oversample = 1
)
```

Arguments

x	numeric vector (or list of numeric vectors). idxTarget and idxControl refer to the elements of x. If x is a list, all elements must have the same length.
idxTarget	numeric or logical vector specifying the elements in x that define the target distribution to be matched by the control elements.
idxControl	numeric or logical vector specifying the complete set of possible control elements in x (default: all that are not in idxTarget), from which a subset is to be sampled.
nbins	numeric(1) or numeric(length(x)) if x is a list, specifying the number of bins to group the values of x into. Higher numbers of bins will increase the match to the target distribution(s), but may fail if there are few elements to sample from (will throw a warning).
oversample	The number of control elements to sample for each target element.

Value

numeric vector with round(length(idxTarget) * oversample) elements, specifying the index (positions) of the sampled control elements.

Author(s)

Michael Stadler

Examples

```
x <- c(runif(1000, min = 0, max = 10),
       rnorm(200, mean = 5, sd = 1))
s <- sampleControlElements(x, idxTarget = 1001:1200, idxControl = 1:1000)
par(mfrow=c(2,2))
h <- hist(x, breaks = 20, main = "all")
hist(x[1:1000], breaks = h$breaks, main = "all control")
hist(x[1001:1200], breaks = h$breaks, main = "target")
hist(x[s], breaks = h$breaks, main = "sampled control")
```

---

selVarGenes *Select Variable Genes in Single Cell RNA-seq*

Description

This function selects the most variable genes from a SingleCellExperiment object using the plot that displays the log2 coefficient of variation as a function of the log2 mean for all genes across all the cells.

Usage

```
selVarGenes(
  data = NULL,
  assay.type = "counts",
  logPseudo = 1,
  Nmads = 3,
  minCells = 5,
  minExpr = 1,
  exclTopExprFrac = 0.01,
  span = 0.2,
  control = stats::loess.control(surface = "direct"),
  nBins = 100,
  nBinsDense = ceiling(nrow(data)/4),
  ...
)
```

Arguments

data	SingleCellExperiment object or normalized count matrix containing the genes as rows and cells as columns.
assay.type	the type of assay to use if data is a SingleCellExperiment. It can be either 'counts' or 'logcounts'. The default is 'counts'.
logPseudo	pseudo-count to use when using the logcounts slot from the SingleCellExperiment to transform back to normalized raw count space.
Nmads	number of MADs beyond which genes are selected per bin.
minCells	keep genes with minimum expression in at least this number of cells.
minExpr	keep genes with expression greater than or equal to this in minCells cells in the normalized count matrix.
exclTopExprFrac	the fraction of top expressed genes that will be excluded from the loess fit (value between 0 and 1).
span	span parameter for loess function.
control	control parameters for loess function.
nBins	number of bins or groups to place the points(genes) into.
nBinsDense	number of bins or groups to use to place the points(genes) into when calculating more accurate distance values to the curve from the loess fit.
...	additional parameters for the loess function from the stats package.

Details

The function takes in a SingleCellExperiment object and calculates the normalized counts by dividing the raw counts by the corresponding sizeFactors per cell, or a matrix of already normalized counts. Only genes that have an expression greater than or equal to minExpr in at least minCells cells will be kept. If assay.type is set to 'logcounts', that assay is transformed back to the raw normalized count space by performing 2^logcounts(data) - 1, under the assumption the logcounts data is in log2 form and had a pseudocount of 1.

The genes that vary most on the log2(coefficient of variation) vs log2(mean) plot of genes will be selected. A loess fit is done on this plot and the distance (euclidean by default) each point has to the curve is calculated in two steps.

In the first step, genes are assigned to bins by taking the minimum distance to the curve. By default we select 100 points on the loess fit and calculate the distances each gene has to all those points on the curve. Each gene is assigned to the point on the curve for which it has the shortest distance. In the second step, more accurate distances to the curve are calculated by using a higher number of points on the curve. Distances are calculated using the `dist.matrix` function.

Finally, for each bin, the most variable genes are selected using the more accurate distance measures. Genes that fall below the loess fit are assigned a negative sign and the genes that are `Nmads` MADs away from the median are selected.

Value

a `list` of length 2:

- varGenes: `vector` containing the names of the most variable genes.
- geneInfo: `data.frame` with genes as rows and columns containing calculated measures:
  - logMean: log2(mean) expression of genes across cells.
  - logCV: log2(coefficient of variation) of genes across cells.
  - pred_logCV: predicted log2(coefficient of variation) from loess fit.
  - assigned_bin: bin each gene has been assigned to.
  - distance: accurate distance measuses. Points below the loess fit get a negative sign.

Author(s)

Dania Machlab

Examples

```
if (requireNamespace("wordspace", quietly = TRUE) &&
    requireNamespace("SingleCellExperiment", quietly = TRUE)) {
    # packages
    library(SingleCellExperiment)

    # create example count matrix
    # ... poisson distr per gene
    mu <- ceiling(runif(n = 2000, min = 0, max = 100))
    counts <- do.call(rbind, lapply(mu, function(x){rpois(1000, lambda = x)}))
    counts <- counts + 1
    # ... add signal to subset of genes (rows) and cells (columns)
    i <- sample(x = 1:nrow(counts), size = 500)
    j <- sample(x = 1:ncol(counts), size = 500)
    counts[i, j] <- counts[i, j] + sample(5:10, length(i), replace = TRUE)

    # create SCE
    sce <- SingleCellExperiment(list(counts = counts))

    # calculate sizeFactors
```

```
    libsizes <- colSums(counts)
    sizeFactors(sce) <- libsizes / mean(libsizes)

    # select variable genes
    varGenes <- selVarGenes(sce, assay.type="counts")

    # plot
    plotSelVarGenes(varGenes, colByBin=TRUE)
    plotSelVarGenes(varGenes)
}
```

---

specificityScore *Calculate gene-expression specificity scores.*

Description

Calculate expression specificity scores for genes that quantify specific expression of a gene in groups of samples (e.g. from different tissues).

Usage

```
specificityScore(
  x,
  method = c("tau", "TSI", "counts"),
  group = NULL,
  thresh = 0,
  expr_values = "logcounts",
  na.rm = FALSE
)

## S4 method for signature 'matrix'
specificityScore(
  x,
  method = c("tau", "TSI", "counts"),
  group = NULL,
  thresh = 0,
  expr_values = "logcounts",
  na.rm = FALSE
)

## S4 method for signature 'SummarizedExperiment'
specificityScore(
  x,
  method = c("tau", "TSI", "counts"),
  group = NULL,
  thresh = 0,
  expr_values = "logcounts",
```

```
  na.rm = FALSE
)
```

Arguments

x	Expression data, either a matrix with expression values for genes (rows) in each sample (columns), or a SummarizedExperiment or SingleCellExperiment object containing such expression data in one of the assays (selected by expr_values).
method	character scalar selecting the type of expression specificity score to be calculated. One of: "tau", "TSI", "counts". See "Details" for method-specific information.
group	character or factor of length ncol(x) that groups the measurements into clusters or tissues, for which expression specificity scores are to be calculated. If NULL (the default), each column of x is assumed to be its own group. If multiple columns belong to the same group, these columns are first aggregated (averaged) before score calculations.
thresh	numeric scalar defining the expression threshold. Values greater than this threshold are interpreted as expressed (used only for some of the methods, see "Details").
expr_values	Integer scalar or string indicating which assay of x contains the expression values, for x of type SummarizedExperiment or SingleCellExperiment. Ignored if x is a matrix.
na.rm	Logical scalar. If TRUE, NA values are excluded in the calculations.

Value

A numeric vector of length nrow(x) with gene scores.

Author(s)

Michael Stadler

References

For a review of tissue-specificity scores, see: "A benchmark of gene expression tissue-specificity metrics" Nadezda Kryuchkova-Mostacci and Marc Robinson-Rechavi Brief Bioinform. 2017 Mar; 18(2): 205–214. doi: 10.1093/bib/bbw008, PMCID: PMC5444245, PMID: 26891983

Examples

```
x <- rbind(g1 = runif(5),
           g2 = c(1, 0, 0, 0, 0),
           g3 = c(.6, .1, .1, .1, .1))
specificityScore(x)
specificityScore(x, method = "TSI")
specificityScore(x, method = "counts", thresh = 0.5)
```

valueToColor *Map numerical values to colors.*

Description

valueToColor takes a numerical vector and maps each value to an R color string.

Usage

```
valueToColor(
  x,
  rng = range(x, na.rm = TRUE),
 col = c("#5E4FA2", "#3288BD", "#66C2A5", "#ABDDA4", "#E6F598", "#FFFFBF", "#FEE08B",
    "#FDAE61", "#F46D43", "#D53E4F", "#9E0142"),
  NA.col = "lightgray",
  alpha = NULL
)
```

Arguments

x	numeric vector with values to be mapped to colors.
rng	numeric(2) giving the range of values to be mapped to colors. By default, this will be the range of finite values in x.
col	vector with R colors defining the palette (must be a valid argument to col2rgb.
NA.col	Single R color to use for NA values in x.
alpha	NULL (default) or numeric(1) between 0 and 255, giving the alpha channel value for the colors (0 = fully transparent, 255 = fully opaque). NULL will use fully opaque colors (alpha = 255). alpha is ignored if col already contain colors with defined alpha values.

Details

The values in [rng[1], rng[2]] will be linearly mapped to the color palette defined by col. Any values in x less (greater) than rng[1] (rng[2]) will be assigned the same color as rng[1] (rng[2]).

Value

A character vector of the same length of x with R colors in hexadecimal string-encoded RGB format.

Author(s)

Michael Stadler

See Also

colorRamp and rgb for the functions called by valueToColor.

Examples

```
x <- rnorm(1000)
y <- rnorm(1000)
cols <- valueToColor(x + y)
plot(x, y, pch = 20, col = cols, main = "default")
```

`weightedMeanByID` *Aggregate different rows assigned to the same ID by calculating a weighted mean*

Description

First row means are calculated to summarize across replicates identified by the groupCol in the colData. Then different row means that are assigned to the same feature ID given by the idCol in the rowData are summarized by calculating a weighted mean. This weighted mean is the sum of the squared row means divided by the sum of the row means. If all row means are 0, they remain 0 in the output.

Usage

```
weightedMeanByID(
  SE,
  assay,
  idCol = "GENEID",
  groupCol = "group",
  log2Transformed = TRUE
)
```

Arguments

`SE`	a `SummarizedExperiment` object that contains an assay with values to be aggregated, a colData column that assigns samples to their group and a rowData column with IDs to indicate which rows to combine.
`assay`	the name of the assay in the SummarizedExperiment object that should be aggregated.
`idCol`	the column name in the rowData of the SummarizedExperiment indicating the feature ID.
`groupCol`	the column name in the colData of the SummarizedExperiment indicating which columns belong to the same group and should be averaged as replicates, before the weighted mean is calculated across rows.
`log2Transformed`	a `logical` indicating whether values in the assay are log2 transformed. If log2Transformed is TRUE, an exponential transformation will be applied before aggregating the values and another log transformation afterwards.

Value

The output is a `data.frame` with one column for each of the unique names in the groupCol and one row for each of the unique IDs in the idCol. The row and column names are the respective unique values. The entries represent the weighted means for each unique feature ID. If all the input values were NA, the aggregated value is also NA, while for all zero, the output remains zero. If log2Transformed is true the output will be log2 transformed again.

Author(s)

Fiona Ross

Examples

```
set.seed(123)
meansRows <- sample(1:100, 10, replace = TRUE)
dat <- unlist(lapply(meansRows, function(m) {
    rnorm(n = 5, mean = m, sd = 0.1*m)
}))
ma <- matrix(dat, nrow = 10, ncol = 5, byrow = TRUE)
IDs <- data.frame(ID = sample(c("A", "B", "C", "D"), size = 10, replace = TRUE))
Groups <- data.frame(group = c("Y","Y", "Z", "Z", "Z"))
mockSE <- SummarizedExperiment::SummarizedExperiment(
                                assays = list(counts = ma),
                                rowData = IDs,
                                colData = Groups)
weightedMeanByID(mockSE, "counts", idCol = "ID", log2Transformed = FALSE)
```

Index