Package ‘monaLisa’

October 13, 2022

Type Package

Title Binned Motif Enrichment Analysis and Visualization

Version 1.2.0

Description Useful functions to work with sequence motifs in the analysis of genomics data. These include methods to annotate genomic regions or sequences with predicted motif hits and to identify motifs that drive observed changes in accessibility or expression. Functions to produce informative visualizations of the obtained results are also provided.

Depends R (>= 4.1)

Imports methods, stats, utils, grDevices, graphics, BiocGenerics, GenomicRanges, TFBSTools, Biostrings, IRanges, stabs, BSgenome, glmnet, S4Vectors, SummarizedExperiment, BiocParallel, grid, circlize, ComplexHeatmap (>= 2.11.1), XVector, GenomeInfoDb, tools, vioplot

Suggests JASPAR2020, BSgenome.Mmusculus.UCSC.mm10, TxDb.Mmusculus.UCSC.mm10.knownGene, knitr, rmarkdown, testthat, BiocStyle, gridExtra

License GPL (>= 3)

Encoding UTF-8

LazyData false

RoxygenNote 7.1.2

VignetteBuilder knitr

biocViews MotifAnnotation, Visualization, FeatureExtraction, Epigenetics

URL https://github.com/fmicompbio/monaLisa, https://bioconductor.org/packages/monaLisa/, https://fmicompbio.github.io/monaLisa/

BugReports https://github.com/fmicompbio/monaLisa/issues

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

git_branch RELEASE_3_15

git_last_commit 2b27273

git_last_commit_date 2022-04-26

Date/Publication 2022-10-13

Author Dania Machlab [aut] (<https://orcid.org/0000-0002-2578-6930>),
Lukas Burger [aut],
Charlotte Soneson [aut] (<https://orcid.org/0000-0003-3833-2169>),
Michael Stadler [aut, cre] (<https://orcid.org/0000-0002-2269-4934>)

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

R topics documented:

`monaLisa-package` *monaLisa - MOtif aNAlysis with Lisa.*

Description

monaLisa is a collection of tools that simplify motif enrichment analyses in genomic regions of interest.

Details

She makes use of her father Homer (http://homer.ucsd.edu/homer/index.html) and other algorithms to search for motif hits and look for enriched motifs in sets of genomic regions, compared to all other regions.

Known motifs can for example be obtained from a collection of transcription factor binding site specificities, such as **JASPAR2020**.

Author(s)

Dania Machlab, Lukas Burger, Charlotte Soneson and Michael Stadler

---

`annoSeqlogo` *Sequence logo annotation*

Description

create an annotation for a `Heatmap` containing sequence logos.

Usage

```
annoSeqlogo(
  grobL,
  which = c("column", "row"),
  space = unit(0.5, "mm"),
  width = NULL,
  height = NULL,
  gp = gpar(fill = NA, col = NA)
)
```

Arguments

`grobL`	A `list` of sequence logo grobs, typically created using `seqLogoGrob`.
`which`	Whether it is a column annotation or a row annotation?
`space`	The space around the image to the annotation grid borders. The value should be a unit object.

`width`	Width of the annotation. The value should be an absolute unit. Width is not allowed to be set for column annotation.
`height`	Height of the annotation. The value should be an absolute unit. Height is not allowed to be set for row annotation.
`gp`	Graphic parameters for annotation grids. Can be used to control the background color in the annotation grids.

Value

An annotation function which can be used in `HeatmapAnnotation`.

Examples

```
if (require(JASPAR2020) && require(TFBSTools) && require(gridExtra)) {
    pfm1 <- getMatrixByID(JASPAR2020, "MA0139")

    g1 <- seqLogoGrob(pfm1)

    anno <- annoSeqlogo(list(g1))
}
```

`bin` *Bin elements of* `x`.

Description

`bin` groups elements of `x` into bins with either a constant number of elements per bin, a constant bin width or according to user-provided bin boundaries.

Usage

```
bin(
  x,
  binmode = c("equalN", "equalWidth", "breaks"),
  nElements = round(length(x)/5),
  nBins = NULL,
  minAbsX = NULL,
  breaks = NULL,
  ...
)
```

Arguments

`x`	A numerical vector with the values used for binning.
`binmode`	The algorithm to be used for binning. Possible values are: "equalN" (default), "equalWidth" or "breaks" (see Details).

`nElements`	The number of elements per bin (only for `binmode="equalN"`). The width of bins is adjusted accordingly.
`nBins`	The number of bins (only for `binmode="equalWidth"`). The number of elements per bin will be variable.
`minAbsX`	The minimal absolute value in `x` for elements to be binned using the `binmode="equalN"` or `binmode="equalWidth"` (ignored for other values of `binmode`). Elements with `x` values in `[-minAbsX,minAbsX]` will be collected in a single bin.
`breaks`	Numerical vector with bin boundaries (only for `binmode="breaks"`). `breaks` has to be ordered and strictly increasing, and has to be of length (number of bins) + 1.
`...`	further arguments to be passed to `cut(x, breaks,include.lowest = TRUE, ...)`, such as `labels=FALSE`.

Details

Elements are binned according to the values in `x` depending on `binmode`:

equalN Items are grouped into a variable number of bins with `nElements` elements each. If `minAbsX` is not `NULL`, elements with `x`-values in `[-minAbsX,minAbsX]` will first be collected in a single bin before binning the remaining elements. The boundaries of this single bin may be slightly adjusted in order to respect the `nElements` elements in the other bins.

equalWidth Items are group into `nBins` bins with a variable number of elements each.

breaks Items are grouped into bins using `cut(x, breaks, include.lowest = TRUE)`

Value

The return value from `cut(x, ...)`, typically a factor of the same length as `x`. Binning mode, bin boundaries and the "neutral" bin are available from `attr(..., "binmode")`, `attr(..., "breaks")` and `attr(..., "bin0")`. For `binmode = "breaks"`, the latter will be `NA`.

See Also

`cut` which is used internally.

Examples

```
set.seed(1)
x <- rnorm(100)
summary(bin(x, "equalN", nElements=10))
summary(bin(x, "equalN", nElements=10, minAbsX=0.5))
summary(bin(x, "equalWidth", nBins=5))
summary(bin(x, "breaks", breaks=c(-10,-1,0,1,10)))
```

`calcBinnedKmerEnr` *Calculate k-mer enrichment in bins of sequences.*

Description

Given a set of sequences and corresponding bins, identify enriched k-mers (n-grams) in each bin. The sequences can be given either directly or as genomic coordinates.

Usage

```
calcBinnedKmerEnr(
  seqs,
  bins = NULL,
  kmerLen = 5,
  background = c("otherBins", "allBins", "zeroBin", "genome", "model"),
  MMorder = 1,
  test = c("fisher", "binomial"),
  includeRevComp = TRUE,
  maxFracN = 0.7,
  maxKmerSize = 3L,
  GCbreaks = c(0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8),
  pseudocount.kmers = 1,
  pseudocount.log2enr = 8,
  p.adjust.method = "BH",
  genome = NULL,
  genome.regions = NULL,
  genome.oversample = 2,
  BPPARAM = SerialParam(),
  verbose = FALSE
)
```

Arguments

`seqs`	`DNAStringSet` object with sequences to test
`bins`	factor of the same length and order as `seqs`, indicating the bin for each sequence. Typically the return value of `bin`. For `background = "genome"` or `background = "model"`, `bins` can be omitted.
`kmerLen`	A `numeric` scalar giving the k-mer length.
`background`	A `character` scalar specifying the background sequences to use. One of `"otherBins"` (default), `"allBins"`, `"zeroBin"`, `"genome"` or `"model"` (see "Details").
`MMorder`	A `numeric` scalar giving the order of the Markov model used to calculate the expected frequencies for `background = "model"`.
`test`	A `character` scalar specifying the type of enrichment test to perform. One of `"fisher"` (default) or `"binomial"`. The enrichment test is one-sided (enriched in foreground).

`includeRevComp`	A `logical` scalar. If `TRUE` (default), count k-mer occurrences in both `seqs` and their reverse-complement, by concatenating `seqs` and their reverse-complemented versions before the counting. This is useful if motifs can be expected to occur on any strand (e.g. DNA sequences of ChIP-seq peaks). If motifs are only expected on the forward strand (e.g. RNA sequences of CLIP-seq peaks), `includeRevComp = FALSE` should be used. Note that `bins` will be recycled for the reverse complemented sequences, which means that each reverse-complemented sequence will be assigned to the same bib as the corresponding forward sequence.
`maxFracN`	A numeric scalar with the maximal fraction of N bases allowed in a sequence (defaults to 0.7). Sequences with higher fractions are excluded from the analysis.
`maxKmerSize`	the maximum k-mer size to consider, when adjusting background sequence weights for k-mer composition compared to the foreground sequences. The default value (3) will correct for mono-, di- and tri-mer composition.
`GCbreaks`	The breaks between GC bins. The default value is based on the hard-coded bins used in Homer.
`pseudocount.kmers`	A `numeric` scalar - will be added to the observed and expected counts for each k-mer to avoid zero values.
`pseudocount.log2enr`	A numerical scalar with the pseudocount to add to foreground and background counts when calculating log2 motif enrichments
`p.adjust.method`	A character scalar selecting the p value adjustment method (used in `p.adjust`).
`genome`	A `BSgenome` or `DNAStringSet` object with the genome sequence. Only used for `background = "genome"` for extracting background sequences.
`genome.regions`	An optional `GRanges` object defining the intervals in `genome` from which background sequences are sampled for `background = "genome"`. If `NULL`, background sequences are sampled randomly from `genome`.
`genome.oversample`	A `numeric` scalar of at least 1.0 defining how many background sequences will be sampled per foreground sequence for `background = "genome"`. Larger values will take longer but improve the sequence composition similarity between foreground and background (see `"Details"`).
`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation.
`verbose`	A `logical` scalar. If `TRUE`, report on progress.

Details

This function implements a binned k-mer enrichment analysis. In each enrichment analysis, the sequences in a specific bin are used as foreground sequences to test for k-mer enrichments comparing to background sequences (defined by `background`, see below), similarly as in done for motifs in `calcBinnedMotifEnrR`. Sequences are weighted to correct for GC and shorter k-mer composition differences between fore- and background sets.

The background sequences are defined according to the value of the `background` argument:

- otherBins: sequences from all other bins (excluding the current bin)
- allBins: sequences from all bins (including the current bin)
- zeroBin: sequences from the "zero bin", defined by the `maxAbsX` argument of `bin`. If `bins` does not define a "zero bin", for example because it was created by `bin(..., maxAbsX = NULL)`, selecting this background definition will abort with an error.
- genome: sequences randomly sampled from the genome (or the intervals defined in `genome.regions` if given). For each foreground sequence, `genome.oversample` background sequences of the same size are sampled (on average). From these, one per foreground sequence is selected trying to match the G+C composition. In order to make the sampling deterministic, a seed number needs to be provided to the `RNGseed` parameter in `SerialParam` or `MulticoreParam` when creating the `BiocParallelParam` instance in `BPPARAM`.
- model: a Markov model of the order `MMorder` is estimated from the foreground sequences and used to estimate expected k-mer frequencies. K-mer enrichments are then calculated comparing observed to these expected frequencies. In order to make the process deterministic, a seed number needs to be provided to the `RNGseed` parameter in `SerialParam` or `MulticoreParam` when creating the `BiocParallelParam` instance in `BPPARAM`.

For each k-mer, the weights of sequences is multiplied with the number of k-mer occurrences in each sequence and summed, separately for foreground (`sumForegroundWgtWithHits`) and background (`sumBackgroundWgtWithHits`) sequences. The function works in ZOOPS (Zero-Or-One-Per-Sequence) mode, so at most one occurrence per sequence is counted, which helps reduce the impact of sequence repeats. The total foreground (`totalWgtForeground`) and background (`totalWgtBackground`) sum of sequence weights is also calculated. If a k-mer has zero `sumForegroundWgtWithHits` and `sumBackgroundWgtWithHits`, then any values (p-values and enrichment) that are calculated using these two numbers are set to NA.

Two statistical tests for the calculation of enrichment log p-value are available: `test = "fisher"` (default) to perform Fisher's exact tests, or `test = "binomial"` to perform binomial tests, using:

- fisher: `fisher.test(x = tab, alternative = "greater")`, where `tab` is the contingency table with the summed weights of sequences in foreground or background sets (rows), and with or without a occurrences of a particular k-mer (columns).
- binomial: `pbinom(q = sumForegroundWgtWithHits - 1, size = totalWgtForeground, prob = sumBackgroundWgtWithHits / totalWgtBackground, lower.tail = FALSE, log.p = TRUE)`

Value

A `SummarizedExperiment` object with motifs in rows and bins in columns, containing seven assays:

- negLog10P: -log10 P values
- negLog10Padj: -log10 adjusted P values
- pearsonResid: k-mer enrichments as Pearson residuals
- expForegroundWgtWithHits: expected number of foreground sequences with motif hits
- log2enr: k-mer enrichments as log2 ratios
- sumForegroundWgtWithHits: Sum of foreground sequence weights in a bin that have k-mer occurrences
- sumBackgroundWgtWithHits: Sum of background sequence weights in a bin that have k-mer occurrences

#' The `rowData` of the object contains annotations (name, PFMs, PWMs and GC fraction) for the k-mers, while the `colData` slot contains summary information about the bins.

See Also

`getKmerFreq` used to calculate k-mer enrichments; `getSeq,BSgenome-method` which is used to extract sequences from genomepkg if `x` is a `GRanges` object; `bplapply` that is used for parallelization; `bin` for binning of regions

Examples

```
seqs <- Biostrings::DNAStringSet(c("GCATGCATGC", "CATGCGCATG"))
bins <- factor(1:2)
calcBinnedKmerEnr(seqs = seqs, bins = bins, kmerLen = 3)
```

`calcBinnedMotifEnrHomer`

Prepare and run HOMER motif enrichment analysis.

Description

Run complete HOMER motif enrichment analysis, consisting of calls to `prepareHomer`, `system2` and `parseHomerOutput`. This function requires `HOMER` to be installed (see `http://homer.ucsd.edu/homer/index.html`) and the path to the tool to be provided (`homerfile` argument).

Usage

```
calcBinnedMotifEnrHomer(
  gr,
  b,
  genomedir,
  outdir,
  motifFile,
  homerfile = findHomer(),
  regionsize = "given",
  pseudocount.log2enr = 8,
  p.adjust.method = "BH",
  Ncpu = 2L,
  verbose = FALSE,
  verbose.Homer = FALSE
)
```

Arguments

`gr`	A `GRanges` object (or an object that can be coerced to one) with the genomic regions to analyze.

`b`	A vector of the same length as `gr` that groups its elements into bins (typically a factor, such as the one returned by `bin`).
`genomedir`	Directory containing sequence files in Fasta format (one per chromosome).
`outdir`	A path specifying the folder into which the output files will be written.
`motifFile`	A file with HOMER formatted PWMs to be used in the enrichment analysis.
`homerfile`	Path and file name of the `findMotifsGenome.pl` HOMER script.
`regionsize`	The peak size to use in HOMER (`"given"` keeps the coordinate region, an integer value will keep only that many bases in the region center).
`pseudocount.log2enr`	A numerical scalar with the pseudocount to add to foreground and background counts when calculating log2 motif enrichments
`p.adjust.method`	A character scalar selecting the p value adjustment method (used in `p.adjust`).
`Ncpu`	Number of parallel threads that HOMER can use.
`verbose`	A logical scalar. If `TRUE`, print progress messages.
`verbose.Homer`	A logical scalar. If `TRUE`, print the console output when running Homer.

Value

A `SummarizedExperiment` object with motifs in rows and bins in columns, containing seven assays:

- negLog10P: -log10 P values
- negLog10Padj: -log10 adjusted P values
- pearsonResid: motif enrichments as Pearson residuals
- expForegroundWgtWithHits: expected number of foreground sequences with motif hits
- log2enr: motif enrichments as log2 ratios
- sumForegroundWgtWithHits: Sum of foreground sequence weights in a bin that have motif hits
- sumBackgroundWgtWithHits: Sum of background sequence weights in a bin that have motif hits

The `rowData` of the object contains annotations (name, PFMs, PWMs and GC fraction) for the motifs, while the `colData` slot contains summary information about the bins.

See Also

The functions that are wrapped: `prepareHomer`, `system2` and `parseHomerOutput`, `bin` for binning of regions

Examples

```
if (!is.na(findHomer())){

  # genome
  genome <-  system.file("extdata", "exampleGenome.fa", package = "monaLisa")
```

```
# create motif file for Homer
motiffile <- tempfile()
motifIDs <- c("MA0139.1", "MA1102.1", "MA0740.1")
dumpJaspar(filename = motiffile, pkg = "JASPAR2020",
           opts = list(ID = motifIDs))

# GRanges of regions used in binned motif enrichment analysis
gr <- GenomicRanges::tileGenome(
    seqlengths = c(chr1 = 10000L, chr2 = 10000L, chr3 = 10000L),
    tilewidth = 200, cut.last.tile.in.chrom = TRUE)

# create bins (motif enrichment analysis will be per bin)
bins <- factor(GenomicRanges::seqnames(gr))
table(bins)

# run calcBinnedMotifEnrHomer
outdir <- tempfile()
se <- calcBinnedMotifEnrHomer(gr = gr, b = bins, genomedir = genome,
    outdir = outdir, motifFile = motiffile)
list.files(outdir)

}
```

`calcBinnedMotifEnrR` *Binned Motif Enrichment Analysis with* `monaLisa`

Description

This function performs a motif enrichment analysis on bins of sequences. For each bin, the sequences in all other bins are used as background.

Usage

```
calcBinnedMotifEnrR(
  seqs,
  bins = NULL,
  pwmL = NULL,
  background = c("otherBins", "allBins", "zeroBin", "genome"),
  test = c("fisher", "binomial"),
  maxFracN = 0.7,
  maxKmerSize = 3L,
  min.score = 10,
  matchMethod = "matchPWM",
  GCbreaks = c(0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8),
  pseudocount.log2enr = 8,
  p.adjust.method = "BH",
  genome = NULL,
  genome.regions = NULL,
```

```
  genome.oversample = 2,
  BPPARAM = SerialParam(),
  verbose = FALSE,
  ...
)
```

Arguments

`seqs`	`DNAStringSet` object with sequences to test
`bins`	factor of the same length and order as `seqs`, indicating the bin for each sequence. Typically the return value of `bin`. For `background = "genome"`, `bins` can be omitted.
`pwmL`	PWMatrixList with motifs for which to calculate enrichments.
`background`	A `character` scalar specifying the background sequences to use. One of `"otherBins"` (default), `"allBins"`, `"zeroBin"` or `"genome"` (see "Details").
`test`	A `character` scalar specifying the type of enrichment test to perform. One of `"fisher"` (default) or `"binomial"`. The enrichment test is one-sided (enriched in foreground).
`maxFracN`	A numeric scalar with the maximal fraction of N bases allowed in a sequence (defaults to 0.7). Sequences with higher fractions are excluded from the analysis.
`maxKmerSize`	the maximum k-mer size to consider, when adjusting background sequence weights for k-mer composition compared to the foreground sequences. The default value (3) will correct for mono-, di- and tri-mer composition.
`min.score`	the minimal score for motif hits, used in `findMotifHits`.
`matchMethod`	the method used to scan for motif hits, passed to the `method` parameter in `findMotifHits`.
`GCbreaks`	The breaks between GC bins. The default value is based on the hard-coded bins used in Homer.
`pseudocount.log2enr`	A numerical scalar with the pseudocount to add to foreground and background counts when calculating log2 motif enrichments
`p.adjust.method`	A character scalar selecting the p value adjustment method (used in `p.adjust`).
`genome`	A `BSgenome` or `DNAStringSet` object with the genome sequence. Only used for `background = "genome"` for extracting background sequences.
`genome.regions`	An optional `GRanges` object defining the intervals in `genome` from which background sequences are sampled for `background = "genome"`. If `NULL`, background sequences are sampled randomly from `genome`.
`genome.oversample`	A `numeric` scalar of at least 1.0 defining how many background sequences will be sampled per foreground sequence for `background = "genome"`. Larger values will take longer but improve the sequence composition similarity between foreground and background (see `"Details"`).

`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation.
`verbose`	A logical scalar. If `TRUE`, print progress messages.
`...`	Additional arguments for `findMotifHits`.

Details

This function implements a binned motif enrichment analysis. In each enrichment analysis, the sequences in a specific bin are used as foreground sequences to test for motif enrichments comparing to background sequences (defined by `background`, see below). The logic follows the `findMotifsGenome.pl` tool from `Homer` version 4.11, with `-size given -nomotif -mknown` and additionally `-h` if using `test = "fisher"`, and gives very similar results. As in the `Homer` tool, sequences are weighted to correct for GC and k-mer composition differences between fore- and background sets.

The background sequences are defined according to the value of the `background` argument:

- otherBins: sequences from all other bins (excluding the current bin)
- allBins: sequences from all bins (including the current bin)
- zeroBin: sequences from the "zero bin", defined by the `maxAbsX` argument of `bin`. If `bins` does not define a "zero bin", for example because it was created by `bin(..., maxAbsX = NULL)`, selecting this background definition will abort with an error.
- genome: sequences randomly sampled from the genome (or the intervals defined in `genome.regions` if given). For each foreground sequence, `genome.oversample` background sequences of the same size are sampled (on average). From these, one per foreground sequence is selected trying to match the G+C composition. In order to make the sampling deterministic, a seed number needs to be provided to the `RNGseed` parameter in `SerialParam` or `MulticoreParam` when creating the `BiocParallelParam` instance in `BPPARAM`.

Motif hits are predicted using `findMotifHits` and multiple hits per sequence are counted as just one hit (ZOOPS mode). For each motif, the weights of sequences that have a hit are summed separately for foreground (`sumForegroundWgtWithHits`) and background (`sumBackgroundWgtWithHits`). The total foreground (`totalWgtForeground`) and background (`totalWgtBackground`) sum of sequence weights is also calculated. If a motif has zero `sumForegroundWgtWithHits` and `sumBackgroundWgtWithHits`, then any values (p-values and enrichment) that are calculated using these two numbers are set to NA.

Two statistical tests for the calculation of enrichment log p-value are available: `test = "fisher"` (default) to perform Fisher's exact tests, or `test = "binomial"` to perform binomial tests (default in `Homer`), using:

- fisher: `fisher.test(x = tab, alternative = "greater")`, where `tab` is the contingency table with the summed weights of sequences in foreground or background sets (rows), and with or without a hit for a particular motif (columns).
- binomial: `pbinom(q = sumForegroundWgtWithHits - 1, size = totalWgtForeground, prob = sumBackgroundWgtWithHits / totalWgtBackground, lower.tail = FALSE, log.p = TRUE)`

Value

A `SummarizedExperiment` object with motifs in rows and bins in columns, containing seven assays:

- negLog10P: -log10 P values
- negLog10Padj: -log10 adjusted P values
- pearsonResid: motif enrichments as Pearson residuals
- expForegroundWgtWithHits: expected number of foreground sequences with motif hits
- log2enr: motif enrichments as log2 ratios
- sumForegroundWgtWithHits: Sum of foreground sequence weights in a bin that have motif hits
- sumBackgroundWgtWithHits: Sum of background sequence weights in a bin that have motif hits

The `rowData` of the object contains annotations (name, PFMs, PWMs and GC fraction) for the motifs, while the `colData` slot contains summary information about the bins.

Examples

```
seqs <- Biostrings::DNAStringSet(c("GTCAGTCGATC", "CAGTCTAGCTG",
                                   "CGATCGTCAGT", "AGCTGCAGTCT"))
bins <- factor(rep(1:2, each = 2))
m <- rbind(A = c(2, 0, 0),
           C = c(1, 1, 0),
           G = c(0, 2, 0),
           T = c(0, 0, 3))
pwms <- TFBSTools::PWMatrixList(
    TFBSTools::PWMatrix(ID = "m1", profileMatrix = m),
    TFBSTools::PWMatrix(ID = "m2", profileMatrix = m[, 3:1])
)
calcBinnedMotifEnrR(seqs = seqs, bins = bins, pwmL = pwms,
                    min.score = 3)
```

`dumpJaspar` *Dump Jaspar motifs into a HOMER motif file.*

Description

Get motifs from a Jaspar database package (e.g. `JASPAR2020`) and write them into a HOMER-compatible motif file as positional probability matrices.

Usage

```
dumpJaspar(
  filename,
  pkg = "JASPAR2020",
  opts = list(tax_group = "vertebrates"),
  pseudocount = 1,
  relScoreCutoff = 0.8,
  verbose = FALSE
)
```

Arguments

`filename`	Name of the output file to be created.
`pkg`	Name of the Jaspar package to use (default: `JASPAR2020`).
`opts`	A list with search options used in `getMatrixSet`. By default, only vertebrate motifs are included in the output using `opts = list(tax_group = "vertebrates")`.
`pseudocount`	A numerical scalar with the pseudocount to be added to each element of the position frequency matrix extracted from Jaspar, before its conversion to a position probability matrix (default: 1.0).
`relScoreCutoff`	Currently ignored. numeric(1) in [0,1] that sets the default motif log-odds score cutof to relScoreCutoff * maximal score for each PWM (default: 0.8).
`verbose`	A logical scalar. If `TRUE`, print progress messages.

Value

`TRUE` if successful.

See Also

`getMatrixSet` for details on the argument `opts`. `homerToPFMatrixList` to read a file with HOMER-formatted motifs into a `PFMatrixList`.

Examples

```
dumpJaspar(filename = tempfile(), pkg = "JASPAR2020",
           opts = list(ID = c("MA0006.1")))
```

`findHomer` *Find HOMER script file.*

Description

Find absolute path to HOMER script file.

Usage

```
findHomer(homerfile = "findMotifsGenome.pl", dirs = NULL)
```

Arguments

`homerfile`	Name of the script file to search.
`dirs`	Directory names to look for `homerfile`. If `dirs=NULL`, all directories listed in the `PATH` environment variable will be searched.

Details

In addition to `dirs`, `findHomer` will also look in the directory provided in the environment variable `MONALISA_HOMER`.

Value

Absolute path to `homerfile`, or `NA` if none or several were found.

Examples

```
homer_path <- findHomer()
```

`findMotifHits` *Find motif matches in sequences.*

Description

`findMotifHits` scans sequences (either provided as a file, an R object or genomic coordinates) for matches to positional weight matrices (provided as a file or as R objects)

Usage

```
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'character,character'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'character,DNAString'
findMotifHits(
  query,
```

```
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'character,DNAStringSet'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'PWMatrix,character'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'PWMatrix,DNAString'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'PWMatrix,DNAStringSet'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
```

```
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'PWMatrixList,character'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'PWMatrixList,DNAString'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'PWMatrixList,DNAStringSet'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)

## S4 method for signature 'PWMatrix,GRanges'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)
```

```
## S4 method for signature 'PWMatrixList,GRanges'
findMotifHits(
  query,
  subject,
  min.score,
  method = c("matchPWM", "homer2"),
  homerfile = findHomer("homer2"),
  BPPARAM = SerialParam(),
  genome = NULL
)
```

Arguments

`query`	The motifs to search for, either a • `character(1)` with the path and file name of a motif file with PWM in HOMER format (currently only supported for `method="homer2"`) • `PWMatrix` with a single PWM • `PWMatrixList` with several PWMs to search for.
`subject`	The sequences to be searched, either a • `character` with the path and file name of a sequence file with DNA sequences in FASTA format • `DNAString` with a single sequence • `DNAStringSet` with several sequences • `GRanges` object with the genomic coordinates of the sequences to be searched.
`min.score`	The minimum score for counting a match. Can be given as a character string containing a percentage (e.g. "85 highest possible score or as a single number.
`method`	The internal method to use for motif searching. One of • `"matchPWM"` using Biostrings::matchPWM (optimized) • `"homer2"` call to the homer2 binary Please note that the two methods might give slightly different results (see details).
`homerfile`	Path and file name of the `homer2` binary.
`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation.
`genome`	`BSgenome` object that is the reference genome of the subject. This argument is set to NULL by default and only used by the function when the subject is a `GRanges` object. It is then necessary to specify the genome so that the function can internally convert the genomic regions into a `DNAStringSet` object.

Details

The implemented methods (`matchPWM` and `homer2`) are there for convenience (`method="matchPWM"` calls `Biostrings::matchPWM` internally in an optimized fashion, and `method = "homer2"` calls the command line tool from Homer and therefore requires an installation of Homer).

In general, running `findMotifHits` with the same parameters using any of the methods generates identical results. Some minor differences could occur that result from rounding errors during the necessary conversion of PWMs (log2-odd scores) to the probability matrices needed by Homer, and the conversion of scores from and to the natural log scale used by Homer. These conversions are implemented transparently for the user, so that the arguments of `findMotifHits` do not have to be adjusted (e.g. the PWMs should always contain log2-odd scores, and `min.score` is always on the log2 scale).

If there are bases with frequencies of less than 0.001 in a motif, Homer will set them to 0.001 and adjust the other frequencies at that motif position accordingly so that they sum to 1.0. This may differ from the adjustment used when scanning a PWM with `matchPWM` (e.g. the `pseudocounts` argument in the `toPWM` function), and thus can give rise to differences in reported motif hits and hit scores (typically only low-scoring hits).

Value

A `GRanges` object with the matches to `query` in `subject`.

Examples

```
seqs <- Biostrings::DNAStringSet(c(s1 = "GTCAGTCGATC", s2 = "CAGTCTAGCTG",
                                   s3 = "CGATCGTCAGT", s4 = "AGCTGCAGTCT"))
m <- rbind(A = c(2, 0, 0),
           C = c(1, 1, 0),
           G = c(0, 2, 0),
           T = c(0, 0, 3))
pwms <- TFBSTools::PWMatrixList(
    TFBSTools::PWMatrix(ID = "m1", profileMatrix = m),
    TFBSTools::PWMatrix(ID = "m2", profileMatrix = m[, 3:1])
)
findMotifHits(pwms, seqs, min.score = 7)
```

`getColsByBin` *Get colors by bin.*

Description

Get colors for elements according to their bin. Colors are assigned to bins forming a gradient from `col1` to `col2` in the order of `levels{b}`. `col0` is assigned to the neutral bin (attribute `""`) if available.

Usage

```
getColsByBin(
  b,
  col1 = c("#003C30", "#01665E", "#35978F", "#80CDC1", "#C7EAE5"),
  col2 = c("#F6E8C3", "#DFC27D", "#BF812D", "#8C510A", "#543005"),
  col0 = "#F5F5F5"
)
```

Arguments

`b`	A factor that groups elements into bins (typically the output of `bin`).
`col1`	First color.
`col2`	Second color.
`col0`	Neutral color.

Value

A character vector with colors for the elements in `b`.

See Also

`bin`.

Examples

```
set.seed(1)
x <- rnorm(100)
b <- bin(x, "equalN", nElements = 10)
cols <- getColsByBin(b)
```

`getKmerFreq` *Calculate observed and expected k-mer frequencies*

Description

Given a set of sequences, calculate observed and expected k-mer frequencies. Expected frequencies are based on a Markov model of order `MMorder`.

Usage

```
getKmerFreq(
  seqs,
  kmerLen = 5,
  MMorder = 1,
  pseudocount = 1,
  zoops = TRUE,
  strata = rep(1L, length(seqs)),
  p.adjust.method = "BH",
  includeRevComp = TRUE
)
```

Arguments

`seqs`	Set of sequences, either a `character` vector or a `DNAStringSet`.
`kmerLen`	A `numeric` scalar giving the k-mer length.
`MMorder`	A `numeric` scalar giving the order of the Markov model used to calculate the expected frequencies.
`pseudocount`	A `numeric` scalar - will be added to the observed counts for each k-mer to avoid zero values.
`zoops`	A `logical` scalar. If `TRUE` (the default), only one or zero occurences of a k-mer are considered per sequence.
`strata`	A `factor` or a `numeric` scalar defining the strata of sequences. A separate Markov model and expected k-mer frequencies are estimated for the set of sequences in each stratum (level in a `strata` factor). If `strata` is a scalar value, it will be interpreted as the number of strata to split the sequences into according to their CpG observed-over-expected counts using `kmeans(CpGoe, centers = strata)`.
`p.adjust.method`	A character scalar selecting the p value adjustment method (used in `p.adjust`).
`includeRevComp`	A `logical` scalar. If `TRUE` (default), count k-mer occurrences in both `seqs` and their reverse-complement, by concatenating `seqs` and their reverse-complemented versions before the counting. This is useful if motifs can be expected to occur on any strand (e.g. DNA sequences of ChIP-seq peaks). If motifs are only expected on the forward strand (e.g. RNA sequences of CLIP-seq peaks), `includeRevComp = FALSE` should be used. Note that if `strata` is a vector of the same length as `seqs`, each reverse-complemented sequence will be assigned to the same stratum as the forward sequence.

Value

A `list` with observed and expected k-mer frequencies (`freq.obs` and `freq.exp`, respectively), and enrichment statistics for each k-mer.

Examples

```
res <- getKmerFreq(seqs = c("AAAAATT", "AAATTTT"), kmerLen = 3)
names(res)
head(res$freq.obs)
head(res$freq.exp)
```

`getSetZeroBin` *Get and set the zero bin manually*

Description

Get and set the zero bin manually

Usage

```
getZeroBin(bins)

setZeroBin(bins, zeroBin)
```

Arguments

`bins`	Factor, typically the return value of `bin`.
`zeroBin`	Numeric or character scalar indicating the level to use as the zero bin, or NA.

Value

For `getZeroBin`, the index of the level representing the zero bin. For `setZeroBin`, a modified factor with the zero bin set to the provided value.

Examples

```
set.seed(1)
x <- rnorm(100)
bins <- bin(x, "equalN", nElements = 10, minAbsX = 0.5)
getZeroBin(bins)
bins <- setZeroBin(bins, 2)
```

`homerToPFMatrixList` *Read a HOMER motif file and create a PFMatrixList*

Description

Read motifs from a file in HOMER format and create a PFMatrixList from them.

Usage

```
homerToPFMatrixList(filename, n = 100L)
```

Arguments

`filename`	Name of the input file with HOMER-formatted motifs.
`n`	The number of observations (multiplied with base frequencies to create the number of observed bases at each position).

Value

A `PFMatrixList` with motifs from the file.

See Also

`dumpJaspar` for writing motifs from a Jaspar database package into a file in HOMER format.

Examples

```
library(JASPAR2020)
optsL <- list(ID = c("MA0006.1"))
pfm1 <- TFBSTools::getMatrixSet(JASPAR2020, opts = optsL)
TFBSTools::Matrix(pfm1)

tmpfn <- tempfile()
dumpJaspar(filename = tmpfn, pkg = "JASPAR2020", opts = optsL)
pfm2 <- homerToPFMatrixList(tmpfn)
TFBSTools::Matrix(pfm2)

unlink(tmpfn)
```

`motifKmerSimilarity` *Calculate similarities between motifs and k-mers.*

Description

For each motif, calculate it's similarity to all k-mers of length `kmerLen`, defined as the maximal probability of observing the k-mer given the base frequencies of the motif (the maximum is taken over for all possible ungapped alignments between motif and k-mer). If necessary matrices are padded on the sides with background base frequencies (assuming all bases to have a frequency of 0.25).

Usage

```
motifKmerSimilarity(
  x,
  kmerLen = 5,
  kmers = NULL,
  includeRevComp = FALSE,
  BPPARAM = SerialParam(),
  verbose = FALSE
)
```

Arguments

`x`	Either a `PFMatrixList`, or a character scalar with a file containing motifs in HOMER format (used directly `method = "HOMER"`, loaded into a `PFMatrixList` by `homerToPFMatrixList` for `method = "R"`).
`kmerLen`	A `numeric` scalar giving the k-mer length.
`kmers`	Either a character vector of k-mers for which to calculate the similarity to each motif, or `NULL`, in which case all k-mers of length `kmerLen` are used.
`includeRevComp`	A `logical` scalar. If set to `TRUE`, each k-mer as well as its reverse complement is compared to each motif, and the larger of the two similarities is returned.

`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation.
`verbose`	A logical scalar. If `TRUE`, report on progress.

Value

A matrix of probabilties for each motif - k-mer pair.

See Also

`bplapply` used for parallelization.

Examples

```
m <- rbind(A = c(12,  0,  0),
           C = c( 3,  2,  0),
           G = c( 0, 14,  0),
           T = c( 0,  0, 15))
pfms <- TFBSTools::PFMatrixList(
    TFBSTools::PFMatrix(name = "m1", profileMatrix = m),
    TFBSTools::PFMatrix(name = "m2", profileMatrix = m[, 3:1])
)
motifKmerSimilarity(pfms, kmerLen = 3)[, c("AGT", "TGA")]
```

`motifSimilarity` *Calculate similarities between pairs of motifs.*

Description

For each pair of motifs, calculate the similarity defined as the maximal Pearson's correlation coefficient between base frequencies over all possible shifts (relative positions of the two matrices with at least one overlapping position). If necessary matrices are padded on the sides with background base frequencies (assuming all bases to have a frequency of 0.25) to enable comparison of all positions in both matrices.

Usage

```
motifSimilarity(
  x,
  y = NULL,
  method = c("R", "HOMER"),
  homerfile = findHomer("compareMotifs.pl"),
  homerOutfile = NULL,
  BPPARAM = SerialParam(),
  verbose = FALSE
)
```

Arguments

`x`	Either a `PFMatrixList`, or a character scalar with a file containing motifs in HOMER format (used directly `method = "HOMER"`, loaded into a `PFMatrixList` by `homerToPFMatrixList` for `method = "R"`).
`y`	Either a `PFMatrixList` or `NULL` (default). If `y = NULL`, then similarities will be calucalted for all pairs of motifs within `x`. Otherwise, `method` must be `"R"` and similarities will be calculated between any motif from `x` to any motif from `y`.
`method`	A character scalar specifying the method for similarity calculations. Either `"R"` (pure R implementation) or `"HOMER"` (will call the `compareMotifs.pl` script from HOMER). Results are identical (apart from rounding errors), and the R implementation is usually faster and can be parallelized (`BPPARAM` argument).
`homerfile`	Path to the HOMER script `compareMotifs.pl` (only used for `method = "HOMER"`.
`homerOutfile`	A character scalar giving the file to save the similarity scores (only for `metho = "HOMER"`). If `NULL`, scores will be stored into a temporary file.
`BPPARAM`	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation (only used for `method = "R"`).
`verbose`	A logical scalar. If `TRUE`, report on progress.

Value

A matrix of Pearson's correlation coefficients for each pair of motifs.

See Also

`bplapply` used for parallelization for `method = "R"`, documentation of HOMER's `compareMotifs.pl` for details on `method = "HOMER"`.

Examples

```
m <- rbind(A = c(12,  0,  0),
           C = c( 3,  2,  0),
           G = c( 0, 14,  0),
           T = c( 0,  0, 15))
pfms <- TFBSTools::PFMatrixList(
    TFBSTools::PFMatrix(name = "m1", profileMatrix = m),
    TFBSTools::PFMatrix(name = "m2", profileMatrix = m + 10),
    TFBSTools::PFMatrix(name = "m3", profileMatrix = m[, 3:1])
)
motifSimilarity(pfms)
```

`parseHomerOutput` *load output from HOMER findMotifsGenome.pl into R*

Description

Parse HOMER output files into R data structures.

Usage

```
parseHomerOutput(infiles, pseudocount.log2enr = 8, p.adjust.method = "BH")
```

Arguments

`infiles`	HOMER output files to be parsed.
`pseudocount.log2enr`	A numerical scalar with the pseudocount to add to foreground and background counts when calculating log2 motif enrichments
`p.adjust.method`	A character scalar selecting the p value adjustment method (used in `p.adjust`).

Value

A list of nine components (`negLog10P`, `negLog10Padj`, `pearsonResid`, `expForegroundWgtWithHits`, `log2enr`, `sumForegroundWgtWithHits` and `sumBackgroundWgtWithHits`), seven containing each a motif (rows) by bin (columns) matrix with raw -log10 P values, -log10 adjusted P values, the expected number of foreground sequences with hits, the observed number of foreground and background sequences with hits, and motif enrichments as Pearson residuals (`pearsonResid`) and as log2 ratios (`log2enr`), and two containing the total foreground and background weight (`totalWgtForeground`, `totalWgtBackground`).

Examples

```
outfile <- system.file("extdata", "homer_output.txt.gz",
                       package = "monaLisa")
res <- parseHomerOutput(infiles = c(bin1 = outfile))
head(res$negLog10P)
```

`plotBinDensity` *Density plot of binned elements.*

Description

Plot the density of binned elements with binning information.

Usage

```
plotBinDensity(
  x,
  b,
  xlab = deparse(substitute(x, env = as.environment(-1))),
  ylab = "Density",
  main = "",
  legend = "topright",
  legend.cex = 1,
  ...
)
```

Arguments

`x`	A numerical vector with the values used for binning.
`b`	A factor that groups elements of `x` into bins (typically the output of `bin`).
`xlab`	Label for x-axis.
`ylab`	Label for y-axis.
`main`	Main title.
`legend`	If not `NULL`, draw a legend with binning information (will be passed to `legend(x=legend)` to control legend position).
`legend.cex`	A scalar that controls the text size in the legend relative to the current `par("cex")` (see `legend`).
`...`	Further arguments passed to `getColsByBin`.

Value

Invisibly the return value of `density(x)` that generated the plot.

See Also

`getColsByBin`

Examples

```
set.seed(1)
x <- rnorm(100)
b <- bin(x, "equalN", nElements = 10)
plotBinDensity(x, b)
```

`plotBinDiagnostics` *Plot diagnostics of binned sequences*

Description

Plot various diagnostics of binned sequences. Three plot types are available:

- `length` plots the distribution of sequence lengths within each bin.
- `GCfrac` plots the distribution of GC fractions within each bin.
- `dinucfreq` plots a heatmap of the relative frequency of each dinucleotide, averaged across the sequences within each bin. The values are centered for each dinucleotide to better highlight differences between the bins. The average relative frequency of each dinucleotide (across the bins) is indicated as well.

Usage

```
plotBinDiagnostics(
  seqs,
  bins,
  aspect = c("length", "GCfrac", "dinucfreq"),
  ...
)
```

Arguments

`seqs`	DNAStringSet object with sequences.
`bins`	factor of the same length and order as seqs, indicating the bin for each sequence. Typically the return value of `bin`.
`aspect`	The diagnostic to plot. Should be one of `"length"`, `"GCfrac"` and `"dinucfreq"`, to plot the distribution of sequence lengths, the distribution of GC fractions and the average relative dinucleotide frequencies across the bins.
`...`	Additional argument passed to `getColsByBin`.

Value

For aspect=`"length"` or `"GCfrac"`, returns (invisibly) the output of `vioplot()`, which generates the plot. For aspect=`"dinucfreq"`, returns (invisibly) the ComplexHeatmap object.

Examples

```
seqs <- Biostrings::DNAStringSet(
  vapply(1:100, function(i) paste(sample(c("A", "C", "G", "T"), 10,
                                        replace = TRUE), collapse = ""), "")
)
bins <- factor(rep(1:2, each = 50))
plotBinDiagnostics(seqs, bins, aspect = "GCfrac")
plotBinDiagnostics(seqs, bins, aspect = "dinucfreq")
```

`plotBinHist` *Histogram of binned elements.*

Description

Plot a histogram of binned elements with binning information.

Usage

```
plotBinHist(
  x,
  b,
  breaks = 10 * nlevels(b),
  xlab = deparse(substitute(x, env = as.environment(-1))),
  ylab = "Frequency",
  main = "",
  legend = "topright",
  legend.cex = 1,
  ...
)
```

Arguments

`x`	A numerical vector with the values used for binning.
`b`	A factor that groups elements of `x` into bins (typically the output of `bin`).
`breaks`	Controls the histogram breaks (passed to `hist(...)`).
`xlab`	Label for x-axis.
`ylab`	Label for y-axis.
`main`	Main title.
`legend`	If not `NULL`, draw a legend with binning information (will be passed to `legend(x=legend)` to control legend position).
`legend.cex`	A scalar that controls the text size in the legend relative to the current `par("cex")` (see `legend`).
`...`	Further arguments passed to `getColsByBin`.

Value

Invisibly the return value of `hist(...)` that generated the plot.

See Also

`getColsByBin`, `hist`

Examples

```
set.seed(1)
x <- rnorm(100)
b <- bin(x, "equalN", nElements = 10)
plotBinHist(x, b)
```

`plotBinScatter` *Scatter plot (xy-plot) of binned elements.*

Description

Plot a scatter (xy-plot) of binned elements with binning information.

Usage

```
plotBinScatter(
  x,
  y,
  b,
  cols = getColsByBin(b),
  xlab = deparse(substitute(x, env = as.environment(-1))),
  ylab = deparse(substitute(y, env = as.environment(-1))),
  main = "",
  legend = "topright",
  legend.cex = 1,
  ...
)
```

Arguments

`x`	A numerical vector with x values.
`y`	A numerical vector with y values (the values used for binning).
`b`	A factor that groups elements of `x,y` into bins (typically the output of `bin(y)`).
`cols`	A color vector (will be computed based on b by default using `getColsByBin(b)`).
`xlab`	Label for x-axis.
`ylab`	Label for y-axis.
`main`	Main title.

`legend`	If not `NULL`, draw a legend with binning information (will be passed to `legend(x=legend)` to control legend position).
`legend.cex`	A scalar that controls the text size in the legend relative to the current `par("cex")` (see `legend`).
`...`	Further arguments passed to `plot(x, y, ...)`.

Value

`TRUE` (invisibly).

See Also

`bin`, `getColsByBin`

Examples

```
set.seed(1)
x <- rnorm(100)
y <- rnorm(100)
b <- bin(y, "equalN", nElements = 10)
plotBinScatter(x, y, b)
```

`plotMotifHeatmaps` *Heatmap of motif enrichments.*

Description

Plot motif enrichments (e.g. significance or magnitude) as a heatmap.

Usage

```
plotMotifHeatmaps(
  x,
  which.plots = c("negLog10P", "pearsonResid", "negLog10Padj", "log2enr"),
  width = 4,
  col.enr = c("#053061", "#2166AC", "#4393C3", "#92C5DE", "#D1E5F0", "#F7F7F7",
    "#FDDBC7", "#F4A582", "#D6604D", "#B2182B", "#67001F"),
  col.sig = c("#F0F0F0", "#D9D9D9", "#BDBDBD", "#969696", "#737373", "#525252",
    "#252525", "#000000"),
  col.gc = c("#F7FCF5", "#E5F5E0", "#C7E9C0", "#A1D99B", "#74C476", "#41AB5D",
    "#238B45", "#006D2C", "#00441B"),
  maxEnr = NULL,
  maxSig = NULL,
  highlight = NULL,
  cluster = FALSE,
  show_dendrogram = FALSE,
```

```
  show_motif_GC = FALSE,
  show_seqlogo = FALSE,
  width.seqlogo = 1.5,
  use_raster = FALSE,
  na_col = "white",
  doPlot = TRUE,
  ...
)
```

Arguments

`x`	A `SummarizedExperiment` with numerical matrices (motifs-by-bins) in its `assays()`, typically the return value of `calcBinnedMotifEnrR` or `calcBinnedMotifEnrHomer`.
`which.plots`	Selects which heatmaps to plot (one or several from `"negLog10P"`, `"negLog10Padj"`, `"pearsonResid"` and `"log2enr"`).
`width`	The width (in inches) of each individual heatmap, without legend.
`col.enr`	Colors used for enrichment heatmap ("pearsonResid" and "log2enr").
`col.sig`	Colors used for significance hetmaps ("negLog10P" and "negLog10Padj").
`col.gc`	Colors used for motif GC content (for `show_motif_GC` = `TRUE`).
`maxEnr`	Cap color mapping at enrichment = `maxEnr` (default: 99.5th percentile).
`maxSig`	Cap color mapping at -log10 P value or -log10 FDR = `maxSig` (default: 99.5th percentile).
`highlight`	A logical vector indicating motifs to be highlighted.
`cluster`	If `TRUE`, the order of transcription factors will be determined by hierarchical clustering of the `"pearsonResid"` component. Alternatively, an `hclust`-object can be supplied which will determine the motif ordering. No reordering is done for `cluster` = `FALSE`.
`show_dendrogram`	If `cluster` != `FALSE`, controls whether to show a row dendrogram for the clustering of motifs. Ignored for `cluster` = `FALSE`.
`show_motif_GC`	If `TRUE`, show a column with the percent G+C of the motif as part of the heatmap.
`show_seqlogo`	If `TRUE`, show a sequence logo next to each motif label. This will likely only make sense for a heatmap with a low number of motifs.
`width.seqlogo`	The width (in inches) for the longest sequence logo (shorter logos are drawn to scale).
`use_raster`	`TRUE` or `FALSE` (default). Passed to `use_raster` of `Heatmap`.
`na_col`	"white" (default). Passed to `na_col` of `Heatmap`.
`doPlot`	If `TRUE` (default), plot the generated heatmap(s) using `Reduce(ComplexHeatmap::add_heatmap, heatmapList)`. If `FALSE`, just return the list of heatmap(s) (`heatmapList`) in example before), allowing to modify them further before plotting.
`...`	Further arguments passed to `Heatmap` when creating the main heatmaps selected by `which.plots`.

Details

The heatmaps are created using the **ComplexHeatmap** package and plotted side-by-side.

Each heatmap will be `width` inches wide, so the total plot needs a graphics device with a width of at least `length(which.plots) * width` plus the space used for motif names and legend. The height will be auto-adjusted to the graphics device.

Value

A list of `ComplexHeatmap::Heatmap` objects.

References

Gu, Z. Complex heatmaps reveal patterns and correlations in multidimensional genomic data. Bioinformatics 2016.

See Also

`bin`, `Heatmap`

Examples

```
se <- readRDS(system.file("extdata",
                          "results.binned_motif_enrichment_LMRs.rds",
                          package = "monaLisa"))
i <- which(SummarizedExperiment::assay(se, "negLog10Padj")[, 8] > 4)
plotMotifHeatmaps(se[i, ], which.plots = "pearsonResid",
                  width = 2, show_seqlogo = TRUE)
```

`plotSelectionProb` *Plot selection probabilities of predictors*

Description

This function plots the selection probabilities of predictors (for example the selected motifs), optionally multiplied with either +1 or -1 to give a sense of both the strength and the directionality of the associated effects. The directionality is estimated from the sign of the correlation coefficient between each predictor and the response vector.

Usage

```
plotSelectionProb(
  se,
  directional = TRUE,
  selProbMin = metadata(se)$stabsel.params.cutoff,
  selProbMinPlot = 0.4,
  showSelProbMin = TRUE,
```

```
  col = c("cadetblue", "grey", "red"),
  method = c("pearson", "kendall", "spearman"),
  ylimext = 0.25,
  legend = "topright",
  legend.cex = 1,
  ...
)
```

Arguments

`se`	The `SummarizedExperiment` object with the results from stability selection (typically returned by `randLassoStabSel`).
`directional`	A logical scalar. If `TRUE`, selection probabilities are plotted with the sign of the marginal correlation between a predictor and the response.
`selProbMin`	A numerical scalar in [0,1]. Predictors with a selection probability greater than `selProbMin` are shown as colored bars. The color is defined by `col[1]`. By default, `selProbMin` is extracted from the parameters stored in `se`.
`selProbMinPlot`	A numerical scalar in [0,1] less than `selProbMin`. Predictors with a selection probability greater than `selProbMinPlot` but less than `selProbMin` are shown as bars with color `col[2]`. `selProbMinPlot` is useful to include additional predictors in the plot that were not selected according to `selProbMin` but may be close to that cutoff. Setting `selProbMinPlot = 0` will create a plot including all predictors.
`showSelProbMin`	A logical scalar. If `TRUE`, the value of `selProbMin` is shown by a horizontal dashed line of color `col[3]`.
`col`	A color vector giving the three colors used for predictors with selection probability greater than `selProbMin`, additional predictors with selection probability greater than `selProbMinPlot`, and the selection probability cutoff line.
`method`	A character scalar with the correlation method to use in the calculation of predictor-response marginal correlations. One of "pearson", "kendall" or "spearman" (see `cor`).
`ylimext`	A numeric scalar defining how much the y axis limits should be expanded beyond the plotted probabilities to allow for space for the bar labels.
`legend`	the position of the legend in the bar plot (will be passed to `legend(x=legend)` to control legend position).
`legend.cex`	A scalar that controls the text size in the legend relative to the current `par("cex")` (see `legend`).
`...`	additional parameters passed to `barplot`.

Details

This function creates a bar plot using the `barplot` function. Each bar corresponds to a predictor (motif) and the colors correspond to whether or not it was selected. The y-axis shows the selection probabilities (`directional=FALSE`) or selection probabilities with the sign of the marginal correlation to the response (`directional=TRUE`).

Value

a `matrix` with one column, containing the coordinates of the bar midpoints, or `NULL` if no bar plot is drawn.

Examples

```
## create data set
Y <- rnorm(n = 500, mean = 2, sd = 1)
X <- matrix(data = NA, nrow = length(Y), ncol = 50)
for (i in seq_len(ncol(X))) {
  X[ ,i] <- runif(n = 500, min = 0, max = 3)
}
s_cols <- sample(x = seq_len(ncol(X)), size = 10,
  replace = FALSE)
for (i in seq_along(s_cols)) {
  X[ ,s_cols[i]] <- X[ ,s_cols[i]] + Y
}

## reproducible randLassoStabSel() with 1 core
set.seed(123)
ss <- randLassoStabSel(x = X, y = Y)
plotSelectionProb(ss)
```

`plotStabilityPaths` *Plot Stability Paths*

Description

Plot the stability paths of each variable (predictor), showing the selection probability as a function of the regularization step.

Usage

```
plotStabilityPaths(
  se,
  selProbMin = metadata(se)$stabsel.params.cutoff,
  col = "cadetblue",
  lwd = 1,
  lty = 1,
  ylim = c(0, 1.1),
  ...
)
```

Arguments

`se`	the `SummarizedExperiment` object resulting from stability selection, by running `randLassoStabSel`.

`selProbMin`	A numerical scalar in [0,1]. Predictors with a selection probability greater than `selProbMin` are shown as colored lines. The color is defined by the `col` argument.
`col`	color of the selected predictors.
`lwd`	line width (default = 1).
`lty`	line type (default = 1).
`ylim`	limits for y-axis (default = c(0,1.1)).
`...`	additional parameters to pass on to `matplot`.

Value

`TRUE` (invisibly).

See Also

`stabsel` and `matplot`

Examples

```
## create data set
Y <- rnorm(n = 500, mean = 2, sd = 1)
X <- matrix(data = NA, nrow = length(Y), ncol = 50)
for (i in seq_len(ncol(X))) {
  X[ ,i] <- runif(n = 500, min = 0, max = 3)
}
s_cols <- sample(x = seq_len(ncol(X)), size = 10,
  replace = FALSE)
for (i in seq_along(s_cols)) {
  X[ ,s_cols[i]] <- X[ ,s_cols[i]] + Y
}

## reproducible randLassoStabSel() with 1 core
set.seed(123)
ss <- randLassoStabSel(x = X, y = Y)
plotStabilityPaths(ss)
```

`prepareHomer` *Prepare input files for HOMER motif enrichment analysis.*

Description

For each bin, write genomic coordinates for foreground and background regions into files for HOMER motif enrichment analysis.

Usage

```
prepareHomer(
  gr,
  b,
  genomedir,
  outdir,
  motifFile,
  homerfile = findHomer(),
  regionsize = "given",
  Ncpu = 2L,
  verbose = FALSE
)
```

Arguments

`gr`	A `GRanges` object (or an object that can be coerced to one) with the genomic regions to analyze.
`b`	A vector of the same length as `gr` that groups its elements into bins (typically a factor).
`genomedir`	Directory containing sequence files in Fasta format (one per chromosome).
`outdir`	A path specifying the folder into which the output files (two files per unique value of `b`) will be written.
`motifFile`	A file with HOMER formatted PWMs to be used in the enrichment analysis.
`homerfile`	Path and file name of the `findMotifsGenome.pl` HOMER script.
`regionsize`	The peak size to use in HOMER (`"given"` keeps the coordinate region, an integer value will keep only that many bases in the region center).
`Ncpu`	Number of parallel threads that HOMER can use.
`verbose`	A logical scalar. If `TRUE`, print progress messages.

Details

For each bin (unique value of `b`) this functions creates two files in `outdir` (`outdir/bin_N_foreground.tab` and `outdir/bin_N_background.tab`, where `N` is the number of the bin and foreground/background correspond to the ranges that are/are not within the current bin). The files are in the HOMER peak file format (see http://homer.ucsd.edu/homer/ngs/peakMotifs.html for details).

In addition, a shell script file is created containing the shell commands to run the HOMER motif enrichment analysis.

Value

The path and name of the script file to run the HOMER motif enrichment analysis.

Examples

```
# prepare genome directory (here: one dummy chromosome)
genomedir <- tempfile()
dir.create(genomedir)
writeLines(c(">chr1", "ATGCATGCATCGATCGATCGATCGTACGTA"),
           file.path(genomedir, "chr1.fa"))

# prepare motif file, regions and bins
motiffile <- tempfile()
dumpJaspar(filename = motiffile, pkg = "JASPAR2020",
           opts = list(ID = c("MA0006.1")))
gr <- GenomicRanges::GRanges("chr1", IRanges::IRanges(1:4, width = 4))
b <- bin(1:4, nElements = 2)

# create dummy file (should point to local Homer installation)
homerfile <- file.path(tempdir(), "findMotifsGenome.pl")
writeLines("dummy", homerfile)

# run prepareHomer
outdir <- tempfile()
prepareHomer(gr = gr, b = b, genomedir = genomedir,
             outdir = outdir, motifFile = motiffile,
             homerfile = homerfile, verbose = TRUE)
list.files(outdir)

# clean up example
unlink(c(genomedir, motiffile, homerfile, outdir))
```

`randLassoStabSel` *Randomized Lasso Stability Selection*

Description

This function runs randomized lasso stability selection as presented by Meinshausen and Bühlmann (2010) and with the improved error bounds introduced by Shah and Samworth (2013). The function uses the `stabsel` function from the `stabs` package, but implements the randomized lasso version.

Usage

```
randLassoStabSel(
  x,
  y,
  weakness = 0.8,
  cutoff = 0.8,
  PFER = 2,
  mc.cores = 1L,
  ...
)
```

Arguments

`x`	the predictor matrix.
`y`	the response vector.
`weakness`	value between 0 and 1 (default = 0.8). It affects how strict the method will be in selecting predictors. The closer it is to 0, the more stringent the selection. A weakness value of 1 is identical to performing lasso stability selection (not the randomized version).
`cutoff`	value between 0 and 1 (default = 0.8) which is the cutoff for the selection probability. Any variable with a selection probability that is higher than the set cutoff will be selected.
`PFER`	integer (default = 2) representing the absolute number of false positives that we allow for in the final list of selected variables. For details see Meinshausen and Bühlmann (2010).
`mc.cores`	integer (default = 1) specifying the number of cores to use in `mclapply`, which is the default way `stabsel` does parallelization.
`...`	additional parameters that can be passed on to `stabsel`.

Details

Randomized lasso stability selection runs a randomized lasso regression several times on subsamples of the response variable and predictor matrix. N/2 elements from the response variable are randomly chosen in each regression, where N is the length of the vector. The corresponding section of the predictor matrix is also chosen, and the internal `.glmnetRandomizedLasso` function is applied. Stability selection results in selection probabilities for each predictor. The probability of a specific predictor is the number of times it was selected divided by the total number of subsamples that were done (total number of times the regression was performed).

We made use of the `stabs` package that implements lasso stability selection, and adapted it to run randomized lasso stability selection.

Value

A `SummarizedExperiment` object where the rows are the observations and the columns the predictors (same dimnames as the predictor matrix `x`). It contains:

- assays:
  - x: the predictor matrix.
- rowData: a DataFrame with columns:
  - y: the response vector.
- colData: a DataFrame with columns:
  - selProb: the final selection probabilities for the predictors (from the last regularization step).
  - selected: logical indicating the predictors that made the selection with the specified cutoff.
  - selAUC: the normalized area under the seletion curve (mean of selection probabilities over regulatization steps).
  - reg'`i`': columns containing the selection probabilities for regularization step i.

- metadata: a list of output returned from `stabsel` and `randLassoStabSel`:
    - stabsel.params.cutoff: probability cutoff set for selection of predictors (see `stabsel`).
    - stabsel.params.selected: elements with maximal selection probability greater `cutoff` (see `stabsel`).
    - stabsel.params.max: maximum of selection probabilities (see `stabsel`).
    - stabsel.params.q: average number of selected variables used (see `stabsel`).
    - stabsel.params.PFER: (realized) upper bound for the per-family error rate (see `stabsel`).
    - stabsel.params.specifiedPFER: specified upper bound for the per-family error rate (see `stabsel`).
    - stabsel.params.p: the number of effects subject to selection (see `stabsel`).
    - stabsel.params.B: the number of subsamples (see `stabsel`).
    - stabsel.params.sampling.type: the sampling type used for stability selection (see `stabsel`).
    - stabsel.params.assumption: the assumptions made on the selection probabilities (see `stabsel`).
    - stabsel.params.call: `stabsel` the call.
    - randStabsel.params.weakness: the weakness parameter in the randomized lasso stability selection.

References

N. Meinshausen and P. Bühlmann (2010), Stability Selection, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **72**, 417–73.
R.D. Shah and R.J. Samworth (2013), Variable Selection with Error Control: Another Look at Stability Selection, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**, 55–80.
B. Hofner, L. Boccuto, and M. Göker (2015), Controlling False Discoveries in High-Dimensional Situations: Boosting with Stability Selection, *BMC Bioinformatics*, **16** 144.

See Also

`stabsel`

Examples

```
## create data set
Y <- rnorm(n = 500, mean = 2, sd = 1)
X <- matrix(data = NA, nrow = length(Y), ncol = 50)
for (i in seq_len(ncol(X))) {
  X[ ,i] <- runif(n = 500, min = 0, max = 3)
}
s_cols <- sample(x = seq_len(ncol(X)), size = 10,
  replace = FALSE)
for (i in seq_along(s_cols)) {
  X[ ,s_cols[i]] <- X[ ,s_cols[i]] + Y
}

## reproducible randLassoStabSel() with 1 core
set.seed(123)
ss <- randLassoStabSel(x = X, y = Y)
```

```
## reproducible randLassoStabSel() in parallel mode
## (only works on non-windows machines)
if (.Platform$OS.type == "unix") {
    RNGkind("L'Ecuyer-CMRG")
    set.seed(123)
    ss <- randLassoStabSel(x = X, y = Y, mc.preschedule = TRUE,
                           mc.set.seed = TRUE, mc.cores = 2L)
}
```

`sampleRandomRegions` *Sample random regions of fixed length.*

Description

Sample random regions from the mappable parts of the genome with a given fraction from CpG islands.

Usage

```
sampleRandomRegions(allowedRegions = NULL, N = 100L, regWidth = 200L)
```

Arguments

`allowedRegions`	An unstranded GRanges object of the "allowed" of the genome, usually the mappable regions.
`N`	Number of regions to sample.
`regWidth`	Region width.

Details

In order to make the results deterministic, set the random number seed before calling `sampleRandomRegions` using `set.seed`.

Value

A GRanges object with randomly sampled mappable regions of width `regWidth` with `fractionCGI` coming from CpG islands.

Examples

```
regs <- GenomicRanges::GRanges(
  seqnames = rep(c("chr1", "chr2"), each = 2),
  ranges = IRanges::IRanges(start = 1:4, end = 5:8))
set.seed(123)
sampleRandomRegions(regs, N = 2, regWidth = 3L)
```

`seqLogoGrob` *Create a simple sequence logo grob.*

Description

Create a simple sequence logo grob (grid-graphics object) for a transcription factor from a position frequency matrix. The logo drawing code is a simplified version from `seqLogo` and for example can be used to embedd sequence logos within other plots.

Usage

```
seqLogoGrob(x, xmax = NULL, ymax = 2, xjust = c("left", "center", "right"))
```

Arguments

`x`	A `PFMatrix` object
`xmax`	A numeric scalar with the maximal width for the logo (in base-pairs). A value of `NULL` will scale the logo to the full width of the viewport.
`ymax`	A numeric scalar with the maximal height for the logo (in bits) A value of `NULL` will scale the logo to the full height of the viewport.
`xjust`	A character scalar specifying the horizontal adjustment of the sequence log withint the viewport; one of `"left"`, `"center"` or `"right"`.

Value

A polygon grob.

See Also

`seqLogo` for the original, more flexible version of this function.

Examples

```
if (require(JASPAR2020) && require(TFBSTools) && require(gridExtra)) {
    pfm1 <- getMatrixByID(JASPAR2020, "MA0139")
    pfm2 <- getMatrixByID(JASPAR2020, "MA0531")

    g1 <- seqLogoGrob(pfm1)
    g2 <- seqLogoGrob(pfm2)

    gridExtra::grid.arrange(g1, g2)
}
```

Index