Package: chromstaR (via r-universe)

July 17, 2024

Type Package

Title Combinatorial and Differential Chromatin State Analysis for ChIP-Seq Data

Version 1.31.0

Author Aaron Taudt, Maria Colome Tatche, Matthias Heinig, Minh Anh Nguyen

Maintainer Aaron Taudt <aaron.taudt@gmail.com>

Description This package implements functions for combinatorial and differential analysis of ChIP-seq data. It includes uni- and multivariate peak-calling, export to genome browser viewable files, and functions for enrichment analyses.

Depends R (>= 3.3), GenomicRanges, ggplot2, chromstaRData

Imports methods, utils, grDevices, graphics, stats, foreach, doParallel, BiocGenerics (>= 0.31.6), S4Vectors, GenomeInfoDb, IRanges, reshape2, Rsamtools, GenomicAlignments, bamsignals, mvtnorm

Suggests knitr, BiocStyle, testthat, biomaRt

URL https://github.com/ataudt/chromstaR

BugReports https://github.com/ataudt/chromstaR/issues

License Artistic-2.0

LazyLoad yes

VignetteBuilder knitr

biocViews ImmunoOncology, Software, DifferentialPeakCalling, HiddenMarkovModel, ChIPSeq, HistoneModification, MultipleComparison, Sequencing, PeakDetection, ATACSeq

RoxygenNote 7.1.0

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

RemoteUrl https://github.com/bioc/chromstaR

RemoteRef HEAD

RemoteSha 3f120edda0081acb2113d5c4f9bc969fdb95d4f7

Contents

`chromstaR-package` *Combinatorial and differential chromatin state analysis for ChIP-seq data*

Description

This package implements functions for the combinatorial and differential analysis of ChIP-seq data. It was developed for histone modifications with a broad profile but is also suitable for the analysis of transcription factor binding data. A Hidden Markov Model with a mixture of Negative Binomials as emission densities is used to call peaks. Please refer to our manuscript at http://dx.doi.org/10.1101/038612 for a detailed description of the method.

Details

The main function of this package is `Chromstar`. For a detailed introduction type `browseVignettes("chromstaR")` and read the vignette. Here is an overview of all `plotting` functions.

Author(s)

Aaron Taudt, Maria Colome-Tatche, Matthias Heinig, Minh Anh Nguyen

---

`binned.data` *Binned read counts*

Description

A `GRanges-class` object which contains binned read counts as meta data column `counts`. It is output of the `binReads` function.

`binReads` *Convert aligned reads from various file formats into read counts in equidistant bins*

Description

Convert aligned reads in .bam or .bed(.gz) format into read counts in equidistant windows.

Usage

```
binReads(
  file,
  experiment.table = NULL,
  ID = NULL,
  assembly,
  bamindex = file,
  chromosomes = NULL,
  pairedEndReads = FALSE,
  min.mapq = 10,
  remove.duplicate.reads = TRUE,
  max.fragment.width = 1000,
  blacklist = NULL,
  binsizes = 1000,
  stepsizes = binsizes/2,
  reads.per.bin = NULL,
  bins = NULL,
  variable.width.reference = NULL,
  use.bamsignals = TRUE,
  format = NULL
)
```

Arguments

`file`	A file with aligned reads. Alternatively a `GRanges-class` with aligned reads.
`experiment.table`	An `experiment.table` containing the supplied `file`. This is necessary to uniquely identify the file in later steps of the workflow. Set to `NULL` if you don't have it (not recommended).

ID	Optional ID to select a row from the experiment.table. Only necessary if the experiment table contains the same file in multiple positions in column 'file'.
assembly	Please see getChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame with columns 'chromosome' and 'length'.
bamindex	BAM index file. Can be specified without the .bai ending. If the index file does not exist it will be created and a warning is issued.
chromosomes	If only a subset of the chromosomes should be binned, specify them here.
pairedEndReads	Set to TRUE if you have paired-end reads in your BAM files (not implemented for BED files).
min.mapq	Minimum mapping quality when importing from BAM files. Set min.mapq=0 to keep all reads.
remove.duplicate.reads	A logical indicating whether or not duplicate reads should be removed.
max.fragment.width	Maximum allowed fragment length. This is to filter out erroneously wrong fragments due to mapping errors of paired end reads.
blacklist	A GRanges-class or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.
binsizes	An integer vector specifying the bin sizes to use.
stepsizes	An integer vector specifying the step size. One number can be given for each element in binsizes, reads.per.bin and bins (in that order).
reads.per.bin	Approximate number of desired reads per bin. The bin size will be selected accordingly.
bins	A GRanges-class or a named list() with GRanges-class containing precalculated bins produced by fixedWidthBins or variableWidthBins. Names of the list must correspond to the binsize. If the list is unnamed, an attempt is made to automatically determine the binsize.
variable.width.reference	A BAM file that is used as reference to produce variable width bins. See variableWidthBins for details.
use.bamsignals	If TRUE the **bamsignals** package is used for parsing of BAM files. This gives tremendous speed advantage for only one binsize but linearly increases for multiple binsizes, while use.bamsignals=FALSE has a binsize dependent runtime and might be faster if many binsizes are calculated.
format	One of c('bed','bam','GRanges',NULL). With NULL the format is determined automatically from the file ending.

Details

Convert aligned reads from .bam or .bed(.gz) files into read counts in equidistant windows (bins). This function uses GenomicRanges::countOverlaps to calculate the read counts, or alternatively bamsignals::bamProfile if option use.bamsignals is set (only effective for .bam files).

Value

If only one bin size was specified for option `binsizes`, the function returns a single GRanges-class object with meta data column 'counts' that contains the read count. If multiple `binsizes` were specified , the function returns a named `list()` of GRanges-class objects.

Examples

```
## Get an example BAM file with ChIP-seq reads
file <- system.file("extdata", "euratrans",
                    "lv-H3K27me3-BN-male-bio2-tech1.bam",
                     package="chromstaRData")
## Bin the file into bin size 1000bp
data(rn4_chrominfo)
data(experiment_table)
binned <- binReads(file, experiment.table=experiment_table,
                   assembly=rn4_chrominfo, binsizes=1000,
                   stepsizes=500, chromosomes='chr12')
print(binned)
```

`callPeaksMultivariate` *Fit a Hidden Markov Model to multiple ChIP-seq samples*

Description

Fit a HMM to multiple ChIP-seq samples to determine the combinatorial state of genomic regions. Input is a list of `uniHMM`s generated by `callPeaksUnivariate`.

Usage

```
callPeaksMultivariate(
  hmms,
  use.states,
  max.states = NULL,
  per.chrom = TRUE,
  chromosomes = NULL,
  eps = 0.01,
  keep.posteriors = FALSE,
  num.threads = 1,
  max.time = NULL,
  max.iter = NULL,
  keep.densities = FALSE,
  verbosity = 1,
  temp.savedir = NULL
)
```

Arguments

hmms	A list of uniHMMs generated by callPeaksUnivariate, e.g. list(hmm1,hmm2,...) or a vector of files that contain such objects, e.g. c("file1","file2",...).
use.states	A data.frame with combinatorial states which are used in the multivariate HMM, generated by function stateBrewer. If both use.states and max.states are NULL, the maximum possible number of combinatorial states will be used.
max.states	Maximum number of combinatorial states to use in the multivariate HMM. The states are ordered by occurrence as determined from the combination of univariate state calls.
per.chrom	If per.chrom=TRUE chromosomes will be treated separately. This tremendously speeds up the calculation but results might be noisier as compared to per.chrom=FALSE, where all chromosomes are concatenated for the HMM.
chromosomes	A vector specifying the chromosomes to use from the models in hmms. The default (NULL) uses all available chromosomes.
eps	Convergence threshold for the Baum-Welch algorithm.
keep.posteriors	If set to TRUE, posteriors will be available in the output. This can be useful to change the posterior cutoff later, but increases the necessary disk space to store the result immensely.
num.threads	Number of threads to use. Setting this to >1 may give increased performance.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default NULL is no limit.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default NULL is no limit.
keep.densities	If set to TRUE (default=FALSE), densities will be available in the output. This should only be needed debugging.
verbosity	Verbosity level for the fitting procedure. 0 - No output, 1 - Iterations are printed.
temp.savedir	A directory where to store intermediate results if per.chrom=TRUE.

Details

Emission distributions from the univariate HMMs are used with a Gaussian copula to generate a multivariate emission distribution for each combinatorial state. This multivariate distribution is then kept fixed and the transition probabilities are fitted with a Baum-Welch. Please refer to our manuscript at http://dx.doi.org/10.1101/038612 for a detailed description of the method.

Value

A multiHMM object.

Author(s)

Aaron Taudt, Maria Colome Tatche

See Also

multiHMM, callPeaksUnivariate, callPeaksReplicates

Examples

```
# Get example BAM files for 2 different marks in hypertensive rat
file.path <- system.file("extdata","euratrans", package='chromstaRData')
files <- list.files(file.path, full.names=TRUE, pattern='SHR.*bam$')[c(1:2,6)]
# Construct experiment structure
exp <- data.frame(file=files, mark=c("H3K27me3","H3K27me3","H3K4me3"),
                condition=rep("SHR",3), replicate=c(1:2,1), pairedEndReads=FALSE,
                controlFiles=NA)
states <- stateBrewer(exp, mode='combinatorial')
# Bin the data
data(rn4_chrominfo)
binned.data <- list()
for (file in files) {
 binned.data[[basename(file)]] <- binReads(file, binsizes=1000, stepsizes=500,
                                           experiment.table=exp,
                                           assembly=rn4_chrominfo, chromosomes='chr12')
}
# Obtain the univariate fits
models <- list()
for (i1 in 1:length(binned.data)) {
 models[[i1]] <- callPeaksUnivariate(binned.data[[i1]], max.time=60, eps=1)
}
# Call multivariate peaks
multimodel <- callPeaksMultivariate(models, use.states=states, eps=1, max.time=60)
# Check some plots
heatmapTransitionProbs(multimodel)
heatmapCountCorrelation(multimodel)
```

---

callPeaksReplicates *Fit a multivariate Hidden Markov Model to multiple ChIP-seq replicates*

Description

Fit an HMM to multiple ChIP-seq replicates and derive correlation measures. Input is a list of uniHMMs generated by callPeaksUnivariate.

Usage

```
callPeaksReplicates(
  hmm.list,
  max.states = 32,
  force.equal = FALSE,
  eps = 0.01,
```

```
  max.iter = NULL,
  max.time = NULL,
  keep.posteriors = TRUE,
  num.threads = 1,
  max.distance = 0.2,
  per.chrom = TRUE
)
```

Arguments

hmm.list	A list of uniHMMs generated by callPeaksUnivariate, e.g. list(hmm1,hmm2,...) or c("file1","file2",...). Alternatively, this parameter also accepts a multiHMM and will check if the distance between replicates is greater than max.distance.
max.states	The maximum number of combinatorial states to consider. The default (32) is sufficient to treat up to 5 replicates exactly and more than 5 replicates approximately.
force.equal	The default (FALSE) allows replicates to differ in their peak-calls, although the majority will usually be identical. If force.equal=TRUE, all peaks will be identical among all replicates.
eps	Convergence threshold for the Baum-Welch algorithm.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default NULL is no limit.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default NULL is no limit.
keep.posteriors	If set to TRUE, posteriors will be available in the output. This is useful to change the post.cutoff later, but increases the necessary disk space to store the result immense.
num.threads	Number of threads to use. Setting this to >1 may give increased performance.
max.distance	This number is used as a cutoff to group replicates based on their distance matrix. The lower this number, the more similar replicates have to be to be grouped together.
per.chrom	If per.chrom=TRUE chromosomes will be treated separately. This tremendously speeds up the calculation but results might be noisier as compared to per.chrom=FALSE, where all chromosomes are concatenated for the HMM.

Value

Output is a multiHMM object with additional entry replicateInfo. If only one uniHMM was given as input, a simple list() with the replicateInfo is returned.

Author(s)

Aaron Taudt

See Also

multiHMM, callPeaksUnivariate, callPeaksMultivariate

Examples

```
# Let's get some example data with 3 replicates
file.path <- system.file("extdata","euratrans", package='chromstaRData')
files <- list.files(file.path, pattern="H3K27me3.*SHR.*bam$", full.names=TRUE)[1:3]
# Obtain chromosome lengths. This is only necessary for BED files. BAM files are
# handled automatically.
data(rn4_chrominfo)
# Define experiment structure
exp <- data.frame(file=files, mark='H3K27me3', condition='SHR', replicate=1:3,
                  pairedEndReads=FALSE, controlFiles=NA)
# We use bin size 1000bp and chromosome 12 to keep the example quick
binned.data <- list()
for (file in files) {
 binned.data[[basename(file)]] <- binReads(file, binsizes=1000, stepsizes=500,
                                           experiment.table=exp,
                                           assembly=rn4_chrominfo, chromosomes='chr12')
}
# The univariate fit is obtained for each replicate
models <- list()
for (i1 in 1:length(binned.data)) {
 models[[i1]] <- callPeaksUnivariate(binned.data[[i1]], max.time=60, eps=1)
}
# Obtain peak calls considering information from all replicates
multi.model <- callPeaksReplicates(models, force.equal=TRUE, max.time=60, eps=1)
```

`callPeaksUnivariate` *Fit a Hidden Markov Model to a ChIP-seq sample.*

Description

Fit a HMM to a ChIP-seq sample to determine the modification state of genomic regions, e.g. call peaks in the sample.

Usage

```
callPeaksUnivariate(
  binned.data,
  control.data = NULL,
  prefit.on.chr = NULL,
  short = TRUE,
  eps = 0.1,
  init = "standard",
  max.time = NULL,
  max.iter = 5000,
```

```
  num.trials = 1,
  eps.try = NULL,
  num.threads = 1,
  read.cutoff = TRUE,
  read.cutoff.quantile = 1,
  read.cutoff.absolute = 500,
  max.mean = Inf,
  post.cutoff = 0.5,
  control = FALSE,
  keep.posteriors = FALSE,
  keep.densities = FALSE,
  verbosity = 1
)
```

Arguments

binned.data	A GRanges-class object with binned read counts or a file that contains such an object.
control.data	Input control for the experiment. A GRanges-class object with binned read counts or a file that contains such an object.
prefit.on.chr	A chromosome that is used to pre-fit the Hidden Markov Model. Set to NULL if you don't want to prefit but use the whole genome instead.
short	If TRUE, the second fitting step is only done with one iteration.
eps	Convergence threshold for the Baum-Welch algorithm.
init	One of the following initialization procedures: standard The negative binomial of state 'unmodified' will be initialized with mean=mean(counts), var=var(counts) and the negative binomial of state 'modified' with mean=mean(counts)+1, var=var(counts). This procedure usually gives the fastest convergence. random Mean and variance of the negative binomials will be initialized with random values (in certain boundaries, see source code). Try this if the 'standard' procedure fails to produce a good fit. empiric Yet another way to initialize the Baum-Welch. Try this if the other two methods fail to produce a good fit.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default NULL is no limit.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default NULL is no limit.
num.trials	The number of trials to run the HMM. Each time, the HMM is seeded with different random initial values. The HMM with the best likelihood is given as output.
eps.try	If code num.trials is set to greater than 1, eps.try is used for the trial runs. If unset, eps is used.
num.threads	Number of threads to use. Setting this to >1 may give increased performance.

`read.cutoff` The default (`TRUE`) enables filtering of high read counts. Set `read.cutoff=FALSE` to disable this filtering.

`read.cutoff.quantile`
A quantile between 0 and 1. Should be near 1. Read counts above this quantile will be set to the read count specified by this quantile. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. If option `read.cutoff.absolute` is also specified, the minimum of the resulting cutoff values will be used. Set `read.cutoff=FALSE` to disable this filtering.

`read.cutoff.absolute`
Read counts above this value will be set to the read count specified by this value. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. If option `read.cutoff.quantile` is also specified, the minimum of the resulting cutoff values will be used. Set `read.cutoff=FALSE` to disable this filtering.

`max.mean` If `mean(counts)>max.mean`, bins with low read counts will be set to 0. This is a workaround to obtain good fits in the case of large bin sizes.

`post.cutoff` False discovery rate. codeNULL means that the state with maximum posterior probability will be chosen, irrespective of its absolute probability (default=codeNULL).

`control` If set to `TRUE`, the binned data will be treated as control experiment. That means only state 'zero-inflation' and 'unmodified' will be used in the HMM.

`keep.posteriors`
If set to `TRUE` (default=`FALSE`), posteriors will be available in the output. This is useful to change the post.cutoff later, but increases the necessary disk space to store the result.

`keep.densities` If set to `TRUE` (default=`FALSE`), densities will be available in the output. This should only be needed debugging.

`verbosity` Verbosity level for the fitting procedure. 0 - No output, 1 - Iterations are printed.

Details

This function is similar to `callPeaksUnivariateAllChr` but allows to pre-fit on a single chromosome instead of the whole genome. This gives a significant performance increase and can help to converge into a better fit in case of unsteady quality for some chromosomes.

Value

A `uniHMM` object.

Author(s)

Aaron Taudt, Maria Colome Tatche

See Also

`uniHMM`, `callPeaksMultivariate`

Examples

```
## Get an example BAM file with ChIP-seq reads
file <- system.file("extdata", "euratrans",
                    "lv-H3K27me3-BN-male-bio2-tech1.bam",
                     package="chromstaRData")
## Bin the BED file into bin size 1000bp
data(rn4_chrominfo)
data(experiment_table)
binned <- binReads(file, experiment.table=experiment_table,
                  assembly=rn4_chrominfo, binsizes=1000,
                  stepsizes=500, chromosomes='chr12')
## Fit the univariate Hidden Markov Model
hmm <- callPeaksUnivariate(binned, max.time=60, eps=1)
## Check if the fit is ok
plotHistogram(hmm)
```

---

callPeaksUnivariateAllChr

Fit a Hidden Markov Model to a ChIP-seq sample.

Description

Fit a HMM to a ChIP-seq sample to determine the modification state of genomic regions, e.g. call peaks in the sample.

Usage

```
callPeaksUnivariateAllChr(
  binned.data,
  control.data = NULL,
  eps = 0.01,
  init = "standard",
  max.time = NULL,
  max.iter = NULL,
  num.trials = 1,
  eps.try = NULL,
  num.threads = 1,
  read.cutoff = TRUE,
  read.cutoff.quantile = 1,
  read.cutoff.absolute = 500,
  max.mean = Inf,
  post.cutoff = 0.5,
  control = FALSE,
  keep.posteriors = FALSE,
  keep.densities = FALSE,
  verbosity = 1
)
```

Arguments

binned.data	A GRanges-class object with binned read counts or a file that contains such an object.
control.data	Input control for the experiment. A GRanges-class object with binned read counts or a file that contains such an object.
eps	Convergence threshold for the Baum-Welch algorithm.
init	One of the following initialization procedures: standard The negative binomial of state 'unmodified' will be initialized with mean=mean(counts), var=var(counts) and the negative binomial of state 'modified' with mean=mean(counts)+1, var=var(counts). This procedure usually gives the fastest convergence. random Mean and variance of the negative binomials will be initialized with random values (in certain boundaries, see source code). Try this if the 'standard' procedure fails to produce a good fit. empiric Yet another way to initialize the Baum-Welch. Try this if the other two methods fail to produce a good fit.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default NULL is no limit.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default NULL is no limit.
num.trials	The number of trials to run the HMM. Each time, the HMM is seeded with different random initial values. The HMM with the best likelihood is given as output.
eps.try	If code num.trials is set to greater than 1, eps.try is used for the trial runs. If unset, eps is used.
num.threads	Number of threads to use. Setting this to >1 may give increased performance.
read.cutoff	The default (TRUE) enables filtering of high read counts. Set read.cutoff=FALSE to disable this filtering.
read.cutoff.quantile	A quantile between 0 and 1. Should be near 1. Read counts above this quantile will be set to the read count specified by this quantile. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. If option read.cutoff.absolute is also specified, the minimum of the resulting cutoff values will be used. Set read.cutoff=FALSE to disable this filtering.
read.cutoff.absolute	Read counts above this value will be set to the read count specified by this value. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. If option read.cutoff.quantile is also specified, the minimum of the resulting cutoff values will be used. Set read.cutoff=FALSE to disable this filtering.

max.mean	If mean(counts)>max.mean, bins with low read counts will be set to 0. This is a workaround to obtain good fits in the case of large bin sizes.
post.cutoff	False discovery rate. codeNULL means that the state with maximum posterior probability will be chosen, irrespective of its absolute probability (default=codeNULL).
control	If set to TRUE, the binned data will be treated as control experiment. That means only state 'zero-inflation' and 'unmodified' will be used in the HMM.
keep.posteriors	If set to TRUE (default=FALSE), posteriors will be available in the output. This is useful to change the post.cutoff later, but increases the necessary disk space to store the result.
keep.densities	If set to TRUE (default=FALSE), densities will be available in the output. This should only be needed debugging.
verbosity	Verbosity level for the fitting procedure. 0 - No output, 1 - Iterations are printed.

Details

The Hidden Markov Model which is used to classify the bins uses 3 states: state 'zero-inflation' with a delta function as emission densitiy (only zero read counts), 'unmodified' and 'modified' with Negative Binomials as emission densities. A Baum-Welch algorithm is employed to estimate the parameters of the distributions. Please refer to our manuscript at http://dx.doi.org/10.1101/038612 for a detailed description of the method.

Value

A uniHMM object.

Author(s)

Aaron Taudt, Maria Coome Tatche

See Also

uniHMM, callPeaksMultivariate

---

changeMaxPostCutoff *Adjust sensitivity of peak detection*

Description

Adjusts the peak calls of a uniHMM, multiHMM or combinedMultiHMM object with a cutoff on the maximum-posterior within each peak. Higher values of maxPost.cutoff mean less sensitive and more precise peak calls. Remaining peaks are kept intact, as opposed to function changePostCutoff, where broad peaks are fragmented. This function was formerly called 'changeFDR' and is still available for backwards compatibiltiy.

Usage

```
changeMaxPostCutoff(model, maxPost.cutoff = 0.99, invert = FALSE)

changeFDR(model, fdr = 0.01, invert = FALSE)
```

Arguments

model	A uniHMM or multiHMM object with posteriors.
maxPost.cutoff	A vector of values between 0 and 1 for each column in model$bins$posteriors. If only one value is given, it will be reused for all columns. Values close to 1 will yield more stringent peak calls with lower false positive but higher false negative rate (i.e. more precise but less sensitive).
invert	Select peaks below (FALSE) or above (TRUE) the given maxPost.cutoff. This is useful to select low confidence peaks.
fdr	Same as 1-maxPost.cutoff.

Details

Each peak has a maximum-posterior (maxPostInPeak, between 0 and 1) associated. The sensitivity is adjusted with a simple cutoff on maxPostInPeak, e.g. for maxPost.cutoff = 0.99 only peaks with maxPostInPeak >= 0.99 will be selected.

Value

The input object is returned with adjusted peak calls.

Functions

- changeFDR: This function was renamed to 'changeMaxPostCutoff' in chromstaR 1.5.1 but it still available for backwards compatibility.

Author(s)

Aaron Taudt

See Also

changePostCutoff

Examples

```
## Get an example uniHMM ##
file <- system.file("data","H3K27me3-BN-rep1.RData", package="chromstaR")
model <- get(load(file))
## Compare fits with different fdrs
plotHistogram(model) + ylim(0,0.25) + ylim(0,0.3)
plotHistogram(changeMaxPostCutoff(model, maxPost.cutoff=0.99)) + ylim(0,0.3)
plotHistogram(changeMaxPostCutoff(model, maxPost.cutoff=1-1e-12)) + ylim(0,0.3)
```

```
## Get an example multiHMM ##
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
genomicFrequencies(model)
model.new <- changeMaxPostCutoff(model, maxPost.cutoff=0.9999, invert=FALSE)
genomicFrequencies(model.new)

## Get an example combinedMultiHMM ##
file <- system.file("data","combined_mode-differential.RData",
                    package="chromstaR")
model <- get(load(file))
genomicFrequencies(model)
model.new <- changeMaxPostCutoff(model, maxPost.cutoff=0.9999, invert=FALSE)
genomicFrequencies(model.new)
```

---

changePostCutoff *Change the posterior cutoff of a Hidden Markov Model*

Description

Adjusts the peak calls of a uniHMM, multiHMM or combinedMultiHMM object with the given posterior cutoff.

Usage

```
changePostCutoff(model, post.cutoff = 0.5)
```

Arguments

model	A uniHMM or multiHMM object with posteriors.
post.cutoff	A vector of posterior cutoff values between 0 and 1 the same length as ncol(model$bins$posteriors). If only one value is given, it will be reused for all columns. Values close to 1 will yield more stringent peak calls with lower false positive but higher false negative rate.

Details

Posterior probabilities are between 0 and 1. Peaks are called if the posteriors for a state (univariate) or sample (multivariate) are >= post.cutoff.

Value

The input object is returned with adjusted peak calls.

Author(s)

Aaron Taudt

See Also

changeMaxPostCutoff

Examples

```
## Get an example BAM file with ChIP-seq reads
file <- system.file("extdata", "euratrans",
                      "lv-H3K27me3-BN-male-bio2-tech1.bam",
                       package="chromstaRData")
## Bin the BED file into bin size 1000bp
data(rn4_chrominfo)
data(experiment_table)
binned <- binReads(file, experiment.table=experiment_table,
                  assembly=rn4_chrominfo, binsizes=1000,
                  stepsizes=500, chromosomes='chr12')
plotHistogram(binned)
## Fit HMM
model <- callPeaksUnivariate(binned, keep.posteriors=TRUE, verbosity=0)
## Compare fits with different post.cutoffs
plotHistogram(changePostCutoff(model, post.cutoff=0.01)) + ylim(0,0.25)
plotHistogram(model) + ylim(0,0.25)
plotHistogram(changePostCutoff(model, post.cutoff=0.99)) + ylim(0,0.25)

## Get an example multiHMM ##
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
genomicFrequencies(model)
model.new <- changePostCutoff(model, post.cutoff=0.9999)
genomicFrequencies(model.new)

## Get an example combinedMultiHMM ##
file <- system.file("data","combined_mode-differential.RData",
                    package="chromstaR")
model <- get(load(file))
genomicFrequencies(model)
model.new <- changePostCutoff(model, post.cutoff=0.9999)
genomicFrequencies(model.new)
```

Chromstar *Wrapper function for the* **chromstaR** *package*

Description

This function performs binning, univariate peak calling and multivariate peak calling from a list of input files.

Usage

```
Chromstar(
  inputfolder,
  experiment.table,
  outputfolder,
  configfile = NULL,
  numCPU = 1,
  binsize = 1000,
  stepsize = binsize/2,
  assembly = NULL,
  chromosomes = NULL,
  remove.duplicate.reads = TRUE,
  min.mapq = 10,
  format = NULL,
  prefit.on.chr = NULL,
  eps.univariate = 0.1,
  max.time = NULL,
  max.iter = 5000,
  read.cutoff.absolute = 500,
  keep.posteriors = TRUE,
  mode = "differential",
  max.states = 128,
  per.chrom = TRUE,
  eps.multivariate = 0.01,
  exclusive.table = NULL
)
```

Arguments

`inputfolder`	Folder with either BAM or BED-6 (see `readBedFileAsGRanges` files.
`experiment.table`	A `data.frame` or tab-separated text file with the structure of the experiment. See `experiment.table` for an example.
`outputfolder`	Folder where the results and intermediate files will be written to.
`configfile`	A file specifying the parameters of this function (without `inputfolder`, `outputfolder` and `configfile`). Having the parameters in a file can be handy if many samples with the same parameter settings are to be run. If a `configfile` is specified, it will take priority over the command line parameters.
`numCPU`	Number of threads to use for the analysis. Beware that more CPUs also means more memory is needed. If you experience crashes of R with higher numbers of this parameter, leave it at `numCPU=1`.
`binsize`	An integer specifying the bin size that is used for the analysis.
`stepsize`	An integer specifying the step size for analysis.
`assembly`	A `data.frame` or tab-separated file with columns 'chromosome' and 'length'. Alternatively a character specifying the assembly, see `getChromInfoFromUCSC` for available assemblies. Specifying an assembly is only necessary when importing BED files. BAM files are handled automatically.

`chromosomes` If only a subset of the chromosomes should be imported, specify them here.

`remove.duplicate.reads`
A logical indicating whether or not duplicate reads should be removed.

`min.mapq` Minimum mapping quality when importing from BAM files. Set `min.mapq=0` to keep all reads.

`format` One of `c('bed','bam',NULL)`. With `NULL` the format is determined automatically from the file ending.

`prefit.on.chr` A chromosome that is used to pre-fit the Hidden Markov Model. Set to `NULL` if you don't want to prefit but use the whole genome instead.

`eps.univariate` Convergence threshold for the univariate Baum-Welch algorithm.

`max.time` The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default `NULL` is no limit.

`max.iter` The maximum number of iterations for the Baum-Welch algorithm. The default `NULL` is no limit.

`read.cutoff.absolute`
Read counts above this value will be set to the read count specified by this value. Filtering very high read counts increases the performance of the Baum-Welch fitting procedure. However, if your data contains very few peaks they might be filtered out. If option `read.cutoff.quantile` is also specified, the minimum of the resulting cutoff values will be used. Set `read.cutoff=FALSE` to disable this filtering.

`keep.posteriors`
If set to `TRUE` (default=`FALSE`), posteriors will be available in the output. This is useful to change the post.cutoff later, but increases the necessary disk space to store the result.

`mode` One of `c('differential','combinatorial','full')`. The modes determine how the multivariate part is run. Here is some advice which mode to use:

- `combinatorial` Each condition is analyzed separately with all marks combined. Choose this mode if you have more than ~7 conditions or you want to have a high sensitivity for detecting combinatorial states. Differences between conditions will be more noisy (more false positives) than in mode `'differential'` but combinatorial states are more precise.
- `differential` Each mark is analyzed separately with all conditions combined. Choose this mode if you are interested in accurate differences. Combinatorial states will be more noisy (more false positives) than in mode `'combinatorial'` but differences are more precise.
- `full` Full analysis of all marks and conditions combined. Best of both, but: Choose this mode only if (number of conditions * number of marks ≤ 8), otherwise it might be too slow or crash due to memory limitations.
- `separate` Only replicates are analyzed multivariately. Combinatorial states are constructed by a simple post-hoc combination of peak calls.

`max.states` The maximum number of states to use in the multivariate part. If set to `NULL`, the maximum number of theoretically possible states is used. CAUTION: This

can be very slow or crash if you have too many states. **chromstaR** has a built in mechanism to select the best states in case that less states than theoretically possible are specified.

`per.chrom` If set to `TRUE` chromosomes will be treated separately in the multivariate part. This tremendously speeds up the calculation but results might be noisier as compared to `per.chrom=FALSE`, where all chromosomes are concatenated for the HMM.

`eps.multivariate`
Convergence threshold for the multivariate Baum-Welch algorithm.

`exclusive.table`
A `data.frame` or tab-separated file with columns 'mark' and 'group'. Histone marks with the same group will be treated as mutually exclusive.

Value

`NULL`

Examples

```
## Prepare the file paths. Exchange this with your input and output directories.
inputfolder <- system.file("extdata","euratrans", package="chromstaRData")
outputfolder <- file.path(tempdir(), 'SHR-example')
## Define experiment structure
data(experiment_table_SHR)
## Define assembly
# This is only necessary if you have BED files, BAM files are handled automatically.
# For common assemblies you can also specify them as 'hg19' for example.
data(rn4_chrominfo)
## Run ChromstaR
Chromstar(inputfolder, experiment.table=experiment_table_SHR,
         outputfolder=outputfolder, numCPU=4, binsize=1000, assembly=rn4_chrominfo,
       prefit.on.chr='chr12', chromosomes='chr12', mode='combinatorial', eps.univariate=1,
         eps.multivariate=1)
```

---

chromstaR-objects *chromstaR objects*

Description

chromstaR defines several objects.

- `uniHMM`: Returned by `callPeaksUnivariate`.
- `multiHMM`: Returned by `callPeaksMultivariate` and `callPeaksReplicates`.
- `combinedMultiHMM`: Returned by `combineMultivariates`.

`collapseBins` *Collapse consecutive bins*

Description

The function will collapse consecutive bins which have, for example, the same combinatorial state.

Usage

```
collapseBins(
  data,
  column2collapseBy = NULL,
  columns2sumUp = NULL,
  columns2average = NULL,
  columns2getMax = NULL,
  columns2drop = NULL
)
```

Arguments

`data`	A data.frame containing the genomic coordinates in the first three columns.
`column2collapseBy`	The number of the column which will be used to collapse all other inputs. If a set of consecutive bins has the same value in this column, they will be aggregated into one bin with adjusted genomic coordinates.
`columns2sumUp`	Column numbers that will be summed during the aggregation process.
`columns2average`	Column numbers that will be averaged during the aggregation process.
`columns2getMax`	Column numbers where the maximum will be chosen during the aggregation process.
`columns2drop`	Column numbers that will be dropped after the aggregation process.

Details

The following tables illustrate the principle of the collapsing:

Input data:

seqnames	start	end	column2collapseBy	moreColumns	columns2sumUp
chr1	0	199	2	1 10	1 3
chr1	200	399	2	2 11	0 3
chr1	400	599	2	3 12	1 3
chr1	600	799	1	4 13	0 3
chr1	800	999	1	5 14	1 3

Output data:

seqnames	start	end	column2collapseBy	moreColumns	columns2sumUp
chr1	0	599	2	1 10	2 9
chr1	600	999	1	4 13	1 6

Value

A data.frame.

Author(s)

Aaron Taudt

Examples

```
## Load example data
## Get an example multiHMM
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
df <- as.data.frame(model$bins)
shortdf <- collapseBins(df, column2collapseBy='state', columns2sumUp='width', columns2average=6:9)
```

---

combinatorialStates *Get the (decimal) combinatorial states of a list of univariate HMM models*

Description

Get the combinatorial states of a list of models generated by `callPeaksUnivariate`. The function returns the decimal combinatorial states for each bin (see details for an explanation of combinatorial state).

Usage

```
combinatorialStates(hmm.list, binary = FALSE)
```

Arguments

`hmm.list`	A list of models generated by `callPeaksUnivariate`, e.g. 'list(model1,model2,...)'.
`binary`	If `TRUE`, a matrix of binary instead of decimal states will be returned.

Details

For a given model, each genomic bin can be either called 'unmodified' or 'modified', depending on the posterior probabilities estimated by the Baum-Welch. Thus, a list of models defines a binary combinatorial state for each bin. This binary combinatorial state can be expressed as a decimal number. Example: We have 4 histone modifications, and we run the univariate HMM for each of them. Then we use a false discovery rate of 0.5 to call each bin either 'unmodified' or 'modified'. The resulting binary combinatorial states can then be converted to decimal representation. The following table illustrates this:

bin	modification state				decimal state
	model1	model2	model3	model4	
1	0	0	1	0	2
2	0	0	0	0	0
3	0	1	1	0	6
4	0	1	1	1	7

Value

Output is a vector of integers representing the combinatorial state of each bin.

Author(s)

Aaron Taudt

See Also

dec2bin, bin2dec

Examples

```
# Get example BAM files for 3 different marks in hypertensive rat (SHR)
file.path <- system.file("extdata","euratrans", package='chromstaRData')
files <- list.files(file.path, full.names=TRUE, pattern='SHR.*bam$')[c(1,4,6)]
# Bin the data
data(rn4_chrominfo)
binned.data <- list()
for (file in files) {
 binned.data[[basename(file)]] <- binReads(file, binsizes=1000, stepsizes=500,
                                           assembly=rn4_chrominfo, chromosomes='chr12')
}
# Obtain the univariate fits
models <- list()
for (i1 in 1:length(binned.data)) {
 models[[i1]] <- callPeaksUnivariate(binned.data[[i1]], max.time=60, eps=1)
}
## Get the decimal representation of the combinatorial state of this combination of models
states <- chromstaR:::combinatorialStates(models, binary=FALSE)
## Show number of each state
table(states)
```

combinedMultiHMM *Combined multivariate HMM object*

Description

The combined multivariate HMM object is output of the function `combineMultivariates` and is a `list()` with various entries. The `class()` attribute of this list was set to "combinedMultiHMM". For a given hmm, the entries can be accessed with the list operators 'hmm[[]]' or 'hmm$'.

Value

A `list()` with the following entries:

`info`	Experiment table for this object.
`bins`	A `GRanges-class` object containing genomic bin coordinates and human readable combinations for the combined `multiHMM` objects.
`segments`	Same as `bins`, but consecutive bins with the same state are collapsed into segments.
`segments.per.condition`	A `list()` with segments for each condition separately.
`peaks`	A `list()` with `GRanges-class` containing peak coordinates for each ID in `info`.
`frequencies`	Genomic frequencies of combinations.
`mode`	Mode of analysis.

See Also

`combineMultivariates`, `uniHMM`, `multiHMM`

combineMultivariates *Combine combinatorial states from several Multivariates*

Description

Combine combinatorial states from several `multiHMM` objects. Combinatorial states can be combined for objects containing multiple marks (`mode='combinatorial'`) or multiple conditions (`mode='differential'`).

Usage

```
combineMultivariates(hmms, mode)
```

Arguments

`hmms`	A `list()` with `multiHMM` objects. Alternatively a character vector with filenames that contain `multiHMM` objects.
`mode`	Mode of combination. See `Chromstar` for a description of the `mode` parameter.

Value

A combinedMultiHMM objects with combinatorial states for each condition.

Author(s)

Aaron Taudt

Examples

```
### Multivariate peak calling for spontaneous hypertensive rat (SHR) ###
# Get example BAM files for 2 different marks in hypertensive rat (SHR)
file.path <- system.file("extdata","euratrans", package='chromstaRData')
files <- list.files(file.path, full.names=TRUE, pattern='SHR.*bam$')[c(1:2,4:5)]
# Construct experiment structure
exp <- data.frame(file=files, mark=c("H3K27me3","H3K27me3","H3K4me3","H3K4me3"),
                condition=rep("SHR",4), replicate=c(1:2,1:2), pairedEndReads=FALSE,
                controlFiles=NA)
states <- stateBrewer(exp, mode='combinatorial')
# Bin the data
data(rn4_chrominfo)
binned.data <- list()
for (file in files) {
 binned.data[[basename(file)]] <- binReads(file, binsizes=1000, stepsizes=500,
                                           experiment.table=exp,
                                           assembly=rn4_chrominfo, chromosomes='chr12')
}
# Obtain the univariate fits
models <- list()
for (i1 in 1:length(binned.data)) {
 models[[i1]] <- callPeaksUnivariate(binned.data[[i1]], max.time=60, eps=1)
}
# Call multivariate peaks
multimodel.SHR <- callPeaksMultivariate(models, use.states=states, eps=1, max.time=60)

#'### Multivariate peak calling for brown norway (BN) rat ###
# Get example BAM files for 2 different marks in brown norway rat
file.path <- system.file("extdata","euratrans", package='chromstaRData')
files <- list.files(file.path, full.names=TRUE, pattern='BN.*bam$')[c(1:2,3:4)]
# Construct experiment structure
exp <- data.frame(file=files, mark=c("H3K27me3","H3K27me3","H3K4me3","H3K4me3"),
                condition=rep("BN",4), replicate=c(1:2,1:2), pairedEndReads=FALSE,
                controlFiles=NA)
states <- stateBrewer(exp, mode='combinatorial')
# Bin the data
data(rn4_chrominfo)
binned.data <- list()
for (file in files) {
 binned.data[[basename(file)]] <- binReads(file, binsizes=1000, stepsizes=500,
                                           experiment.table=exp,
                                           assembly=rn4_chrominfo, chromosomes='chr12')
}
# Obtain the univariate fits
```

```
models <- list()
for (i1 in 1:length(binned.data)) {
 models[[i1]] <- callPeaksUnivariate(binned.data[[i1]], max.time=60, eps=1)
}
# Call multivariate peaks
multimodel.BN <- callPeaksMultivariate(models, use.states=states, eps=1, max.time=60)

### Combine multivariates ###
hmms <- list(multimodel.SHR, multimodel.BN)
comb.model <- combineMultivariates(hmms, mode='combinatorial')
```

`conversion` *Conversion of decimal and binary states*

Description

Convert combinatorial states in decimal representation to combinatorial states in binary representation and vice versa.

Usage

```
dec2bin(dec, colnames = NULL, ndigits = NULL)

bin2dec(bin)
```

Arguments

`dec`	A vector with whole numbers.
`colnames`	The column names for the returned matrix. If specified, `ndigits` will be the length of `colnames`.
`ndigits`	The number of digits that the binary representation should have. If unspecified, the shortest possible representation will be chosen.
`bin`	A matrix with only 0 and 1 (or TRUE and FALSE) as entries. One combinatorial state per row.

Details

chromstaR uses decimal numbers to represent combinatorial states of peaks. These functions serve as a convenient way to get from the efficient decimal representation to a more human-readable binary representation.

Value

A vector of integers for `bin2dec` and a matrix of logicals with one state per row for `dec2bin`.

Functions

- dec2bin: Decimal to binary conversion.
- bin2dec: Binary to decimal conversion.

Author(s)

Aaron Taudt

Examples

```
decimal.states <- c(0:31)
binary.states <- dec2bin(decimal.states, colnames=paste0('mark',1:5))
control.decimal.states <- bin2dec(binary.states)
```

---

enrichmentAtAnnotation

Enrichment of (combinatorial) states for genomic annotations

Description

The function calculates the enrichment of a genomic feature with peaks or combinatorial states. Input is a multiHMM object (containing the peak calls and combinatorial states) and a GRanges-class object containing the annotation of interest (e.g. transcription start sites or genes).

Usage

```
enrichmentAtAnnotation(
  bins,
  info,
  annotation,
  bp.around.annotation = 10000,
  region = c("start", "inside", "end"),
  what = "combinations",
  num.intervals = 21,
  statistic = "fold"
)
```

Arguments

bins	The $bins entry from a multiHMM or combinedMultiHMM object.
info	The $info entry from a multiHMM or combinedMultiHMM object.
annotation	A GRanges-class object with the annotation of interest.
bp.around.annotation	An integer specifying the number of basepairs up- and downstream of the annotation for which the enrichment will be calculated.

region	A combination of c('start','inside','end') specifying the region of the annotation for which the enrichment will be calculated. Select 'start' if you have a point-sized annotation like transcription start sites. Select c('start','inside','end') if you have long annotations like genes.
what	One of c('combinations','peaks','counts') specifying on which feature the statistic is calculated.
num.intervals	Number of intervals for enrichment 'inside' of annotation.
statistic	The statistic to calculate. Either 'fold' for fold enrichments or 'fraction' for fraction of bins falling into the annotation.

Value

A list() containing data.frame()s for enrichment of combinatorial states and binary states at the start, end and inside of the annotation.

Author(s)

Aaron Taudt

---

enrichment_analysis *Enrichment analysis*

Description

Plotting functions for enrichment analysis of multiHMM or combinedMultiHMM objects with any annotation of interest, specified as a GRanges-class object.

Usage

```
plotFoldEnrichHeatmap(
  hmm,
  annotations,
  what = "combinations",
  combinations = NULL,
  marks = NULL,
  plot = TRUE,
  logscale = TRUE
)

plotEnrichCountHeatmap(
  hmm,
  annotation,
  bp.around.annotation = 10000,
  max.rows = 1000,
  combinations = NULL,
  colorByCombinations = sortByCombinations,
```

```
  sortByCombinations = is.null(sortByColumns),
  sortByColumns = NULL
)

plotEnrichment(
  hmm,
  annotation,
  bp.around.annotation = 10000,
  region = c("start", "inside", "end"),
  num.intervals = 20,
  what = "combinations",
  combinations = NULL,
  marks = NULL,
  statistic = "fold",
  logscale = TRUE
)
```

Arguments

hmm	A combinedMultiHMM or multiHMM object or a file that contains such an object.
annotations	A list() with GRanges-class objects containing coordinates of multiple annotations The names of the list entries will be used to name the return values.
what	One of c('combinations','peaks','counts','transitions') specifying on which feature the statistic is calculated.
combinations	A vector with combinations for which the enrichment will be calculated, e.g. combinations = c('[H3K4me3]','[H3K4me3+H3K27me3]'). If NULL all combinations will be considered.
marks	A vector with marks for which the enrichment is plotted. If NULL all marks will be considered.
plot	A logical indicating whether the plot or an array with the fold enrichment values is returned.
logscale	Set to TRUE to plot fold enrichment on log-scale. Ignored if statistic = 'fraction'.
annotation	A GRanges-class object with the annotation of interest.
bp.around.annotation	An integer specifying the number of basepairs up- and downstream of the annotation for which the enrichment will be calculated.
max.rows	An integer specifying the number of randomly subsampled rows that are plotted from the annotation object. This is necessary to avoid crashing for heatmaps with too many rows.
colorByCombinations	A logical indicating whether or not to color the heatmap by combinations.
sortByCombinations	A logical indicating whether or not to sort the heatmap by combinations.
sortByColumns	An integer vector specifying the column numbers by which to sort the rows. If sortByColumns is specified, will force sortByCombinations=FALSE.

region	A combination of c('start','inside','end') specifying the region of the annotation for which the enrichment will be calculated. Select 'start' if you have a point-sized annotation like transcription start sites. Select c('start','inside','end') if you have long annotations like genes.
num.intervals	Number of intervals for enrichment 'inside' of annotation.
statistic	The statistic to calculate. Either 'fold' for fold enrichments or 'fraction' for fraction of bins falling into the annotation.

Value

A ggplot object containing the plot or a list() with ggplot objects if several plots are returned. For plotFoldEnrichHeatmap a named array with fold enrichments if plot=FALSE.

Functions

- plotFoldEnrichHeatmap: Compute the fold enrichment of combinatorial states for multiple annotations.
- plotEnrichCountHeatmap: Plot read counts around annotation as heatmap.
- plotEnrichment: Plot fold enrichment of combinatorial states around and inside of annotation.

Author(s)

Aaron Taudt

See Also

plotting

Examples

```
### Get an example multiHMM ###
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))

### Obtain gene coordinates for rat from biomaRt ###
library(biomaRt)
ensembl <- useEnsembl(biomart='ENSEMBL_MART_ENSEMBL', dataset='rnorvegicus_gene_ensembl')
genes <- getBM(attributes=c('ensembl_gene_id', 'chromosome_name', 'start_position',
                            'end_position', 'strand', 'external_gene_name',
                            'gene_biotype'),
               mart=ensembl)
# Transform to GRanges for easier handling
genes <- GRanges(seqnames=paste0('chr',genes$chromosome_name),
                 ranges=IRanges(start=genes$start, end=genes$end),
                 strand=genes$strand,
                 name=genes$external_gene_name, biotype=genes$gene_biotype)
# Rename chrMT to chrM
seqlevels(genes)[seqlevels(genes)=='chrMT'] <- 'chrM'
```

```
print(genes)

### Make the enrichment plots ###
# We expect promoter [H3K4me3] and bivalent-promoter signatures [H3K4me3+H3K27me3]
# to be enriched at transcription start sites.
   plotEnrichment(hmm = model, annotation = genes, bp.around.annotation = 15000) +
   ggtitle('Fold enrichment around genes') +
   xlab('distance from gene body')

# Plot enrichment only at TSS. We make use of the fact that TSS is the start of a gene.
   plotEnrichment(model, genes, region = 'start') +
   ggtitle('Fold enrichment around TSS') +
   xlab('distance from TSS in [bp]')
# Note: If you want to facet the plot because you have many combinatorial states you
# can do that with
   plotEnrichment(model, genes, region = 'start') +
   facet_wrap(~ combination)

# Another form of visualization that shows every TSS in a heatmap
# If transparency is not supported try to plot to pdf() instead.
   tss <- resize(genes, width = 3, fix = 'start')
   plotEnrichCountHeatmap(model, tss) +
   theme(strip.text.x = element_text(size=6))

# Fold enrichment with different biotypes, showing that protein coding genes are
# enriched with (bivalent) promoter combinations [H3K4me3] and [H3K4me3+H3K27me3],
# while rRNA is enriched with the empty [] and repressive combinations [H3K27me3].
   tss <- resize(genes, width = 3, fix = 'start')
   biotypes <- split(tss, tss$biotype)
   plotFoldEnrichHeatmap(model, annotations=biotypes) + coord_flip()
```

`experiment.table` *Experiment data table*

Description

A `data.frame` specifying the structure of the experiment.

Format

A `data.frame` with columns 'file', 'mark', 'condition', 'replicate', 'pairedEndReads' and 'controlFiles'. Avoid the use of special characters like '-' or '+' as this will confuse the internal file management.

Examples

```
data(experiment_table)
print(experiment_table)
```

exportFiles *Export genome browser uploadable files*

Description

These functions allow to export chromstaR-objects as files which can be uploaded to a genome browser. Peak calls are exported in BED format (.bed.gz), read counts in wiggle format (.wig.gz) as RPKM values, and combinatorial states are exported in BED format (.bed.gz).

Usage

```
exportPeaks(
  model,
  filename,
  header = TRUE,
  separate.files = TRUE,
  trackname = NULL
)

exportCounts(
  model,
  filename,
  header = TRUE,
  separate.files = TRUE,
  trackname = NULL
)

exportCombinations(
  model,
  filename,
  header = TRUE,
  separate.files = TRUE,
  trackname = NULL,
  exclude.states = "[]",
  include.states = NULL
)
```

Arguments

model	A chromstaR-objects.
filename	The name of the file that will be written. The appropriate ending will be appended, either "_peaks.bed.gz" for peak-calls or "_counts.wig.gz" for read counts or "_combinations.bed.gz" for combinatorial states. Any existing file will be overwritten.
header	A logical indicating whether the output file will have a heading track line (TRUE) or not (FALSE).

separate.files	A logical indicating whether or not to produce separate files for each track.
trackname	Name that will be used in the "track name" field of the BED file.
exclude.states	A character vector with combinatorial states that will be excluded from export.
include.states	A character vector with combinatorial states that will be exported. If specified, exclude.states is ignored.

Value

NULL

Functions

- exportPeaks: Export peak calls in BED format.
- exportCounts: Export read counts as RPKM values in wiggle format.
- exportCombinations: Export combinatorial states in BED format.

Examples

```
## Get an example multiHMM
file <- system.file("data","combined_mode-differential.RData",
                    package="chromstaR")
model <- get(load(file))
## Export peak calls and combinatorial states
exportPeaks(model, filename=tempfile())
exportCombinations(model, filename=tempfile())
```

---

exportGRangesAsBedFile

Export genome browser viewable files

Description

Export GRanges as genome browser viewable file

Usage

```
exportGRangesAsBedFile(
  gr,
  trackname,
  filename,
  namecol = "combination",
  scorecol = "score",
  colorcol = NULL,
  colors = NULL,
  header = TRUE,
  append = FALSE
)
```

Arguments

gr	A GRanges-class object.
trackname	The name that will be used as track name and description in the header.
filename	The name of the file that will be written. The ending ".bed.gz". Any existing file will be overwritten.
namecol	A character specifying the column that is used as name-column.
scorecol	A character specifying the column that is used as score-column. The score should contain integers in the interval [0,1000] for compatibility with the UCSC genome browser convention.
colorcol	A character specifying the column that is used for coloring the track. There will be one color for each unique element in colorcol.
colors	A character vector with the colors that are used for the unique elements in colorcol.
header	A logical indicating whether the output file will have a heading track line (TRUE) or not (FALSE).
append	Whether or not to append to an existing file.

Details

Export regions from GRanges-class as a file which can be uploaded into a genome browser. Regions are exported in BED format (.bed.gz).

Value

NULL

Author(s)

Aaron Taudt

See Also

exportPeaks, exportCounts, exportCombinations

Examples

```
### Export regions with read counts above 20 ###
# Get an example BAM file with ChIP-seq reads
file <- system.file("extdata", "euratrans",
                    "lv-H3K27me3-BN-male-bio2-tech1.bam",
                     package="chromstaRData")
# Bin the file into bin size 1000bp
data(rn4_chrominfo)
binned <- binReads(file, assembly=rn4_chrominfo, binsizes=1000,
                 stepsizes=500, chromosomes='chr12')
plotHistogram(binned)
# Export regions with read count above 20
exportGRangesAsBedFile(binned[binned$counts[,1] > 20], filename=tempfile(),
```

```
                trackname='read counts above 20')
```

`fixedWidthBins` *Make fixed-width bins*

Description

Make fixed-width bins based on given bin size.

Usage

```
fixedWidthBins(
  bamfile = NULL,
  assembly = NULL,
  chrom.lengths = NULL,
  chromosome.format,
  binsizes = 1e+06,
  chromosomes = NULL
)
```

Arguments

`bamfile`	A BAM file from which the header is read to determine the chromosome lengths. If a `bamfile` is specified, option `assembly` is ignored.
`assembly`	An assembly from which the chromosome lengths are determined. Please see `getChromInfoFromUCSC` for available assemblies. This option is ignored if `bamfile` is specified. Alternatively a data.frame generated by `getChromInfoFromUCSC`.
`chrom.lengths`	A named character vector with chromosome lengths. Names correspond to chromosomes.
`chromosome.format`	A character specifying the format of the chromosomes if `assembly` is specified. Either 'NCBI' for (1,2,3 ...) or 'UCSC' for (chr1,chr2,chr3 ...). If a `bamfile` or `chrom.lengths` is supplied, the format will be chosen automatically.
`binsizes`	A vector of bin sizes in base pairs.
`chromosomes`	A subset of chromosomes for which the bins are generated.

Value

A `list()` of `GRanges-class` objects with fixed-width bins.

Author(s)

Aaron Taudt

Examples

```
## Make fixed-width bins of size 500kb and 1Mb
data(rn4_chrominfo)
chrom.lengths <- rn4_chrominfo$length
names(chrom.lengths) <- rn4_chrominfo$chromosome
bins <- fixedWidthBins(chrom.lengths=chrom.lengths, binsizes=c(5e5,1e6))
bins

## Make bins using NCBI server (requires internet connection)
# bins <- fixedWidthBins(assembly='mm10', chromosome.format='NCBI', binsizes=c(5e5,1e6))
```

---

genes_rn4 *Gene coordinates for rn4*

Description

A data.frame containing gene coordinates and biotypes of the rn4 assembly.

Format

A data.frame.

Examples

```
data(genes_rn4)
head(genes_rn4)
```

---

genomicFrequencies *Frequencies of combinatorial states*

Description

Get the genomewide frequency of each combinatorial state.

Usage

```
genomicFrequencies(multi.hmm, combinations = NULL, per.mark = FALSE)
```

Arguments

multi.hmm	A multiHMM or combinedMultiHMM object or a file that contains such an object.
combinations	A vector with combinations for which the frequency will be calculated. If NULL all combinations will be considered.
per.mark	Set to TRUE if you want frequencies per mark instead of per combination.

Value

A table with frequencies of each combinatorial state.

Author(s)

Aaron Taudt

Examples

```
## Get an example multiHMM
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
genomicFrequencies(model)
```

`getCombinations` *Get combinations*

Description

Get a DataFrame with combinations from a `GRanges-class` object.

Usage

```
getCombinations(gr)
```

Arguments

`gr`	A `GRanges-class` object from which the meta-data columns containing combinations will be extracted.

Value

A DataFrame.

Examples

```
### Get an example multiHMM ###
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
### Get the combinations
bin.combs <- getCombinations(model$bins)
print(bin.combs)
seg.combs <- getCombinations(model$segments)
print(seg.combs)
```

getDistinctColors *Get distinct colors*

Description

Get a set of distinct colors selected from `colors`.

Usage

```
getDistinctColors(
  n,
  start.color = "blue4",
  exclude.colors = c("white", "black", "gray", "grey", "\\<yellow\\>", "yellow1",
    "lemonchiffon"),
  exclude.brightness.above = 1,
  exclude.rgb.above = 210
)
```

Arguments

`n`	Number of colors to select. If `n` is a character vector, `length(n)` will be taken as the number of colors and the colors will be named by `n`.
`start.color`	Color to start the selection process from.
`exclude.colors`	Character vector with colors that should not be used.
`exclude.brightness.above`	Exclude colors where the 'brightness' value in HSV space is above. This is useful to obtain a matt palette.
`exclude.rgb.above`	Exclude colors where all RGB values are above. This is useful to exclude whitish colors.

Details

The function computes the euclidian distance between all `colors` and iteratively selects those that have the furthest closes distance to the set of already selected colors.

Value

A character vector with colors.

Author(s)

Aaron Taudt

Examples

```
cols <- getDistinctColors(5)
pie(rep(1,5), labels=cols, col=cols)
```

---

getStateColors *Get state colors*

Description

Get the colors that are used for plotting.

Usage

```
getStateColors(labels = NULL)
```

Arguments

labels	Any combination of `c("zero-inflation","unmodified","modified", "total", "counts")`.

Value

A character vector with colors.

See Also

plotting

Examples

```
cols <- getStateColors()
pie(1:length(cols), col=cols, labels=names(cols))
```

---

heatmapCombinations *Plot a heatmap of combinatorial states*

Description

Plot a heatmap that shows the binary presence/absence of marks for the different combinations.

Usage

```
heatmapCombinations(model = NULL, marks = NULL, emissionProbs = NULL)
```

Arguments

model	A multiHMM object or file that contains such an object.
marks	A character vector with histone marks. If specified, model will be ignored.
emissionProbs	A matrix with emission probabilities where dimnames(emissionProbs) gives the state labels and marks. This option is helpful to plot probabilistic chromatin states (not part of **chromstaR**). If specified, model and marks will be ignored.

Value

A ggplot object.

Author(s)

Aaron Taudt

See Also

plotting

Examples

```
marks <- c('H3K4me3','H3K27me3','H4K20me1')
heatmapCombinations(marks=marks)

file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
heatmapCombinations(file)
```

---

heatmapCountCorrelation

Read count correlation heatmap

Description

Heatmap of read count correlations (see cor).

Usage

```
heatmapCountCorrelation(model, cluster = TRUE)
```

Arguments

model	A multiHMM or combinedMultiHMM object or file that contains such an object.
cluster	Logical indicating whether or not to cluster the heatmap.

Value

A ggplot object.

See Also

plotting

Examples

```
## Get an example multiHMM ##
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
## Plot count correlations as heatmap
heatmapCountCorrelation(model)
```

---

heatmapTransitionProbs

Heatmap of transition probabilities

Description

Plot a heatmap of transition probabilities for a multiHMM model.

Usage

```
heatmapTransitionProbs(
  model = NULL,
  reorder.states = TRUE,
  transitionProbs = NULL
)
```

Arguments

model	A multiHMM object or file that contains such an object.
reorder.states	Whether or not to reorder the states.
transitionProbs	A matrix with transition probabilities where dimnames(emissionProbs) gives the state labels. This option is helpful to plot transition probabilities directly without needing a chromstaR-objects. If specified, model will be ignored.

Value

A ggplot object.

See Also

plotting

Examples

```
## Get an example multiHMM ##
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
## Plot transition probabilites as heatmap
heatmapTransitionProbs(model, reorder.states=TRUE)
```

---

loadHmmsFromFiles *Load* **chromstaR** *objects from file*

Description

Wrapper to load **chromstaR** objects from file and check the class of the loaded objects.

Usage

```
loadHmmsFromFiles(
  files,
  check.class = c("GRanges", "uniHMM", "multiHMM", "combinedMultiHMM")
)
```

Arguments

files	A list of chromstaR-objects or a vector of files that contain such objects.
check.class	Any combination of c('GRanges', 'uniHMM', 'multiHMM', 'combinedMultiHMM'). If any of the loaded objects does not belong to the specified class, an error is thrown.

Value

A list of chromstaR-object.

Examples

```
## Get an example BAM file
file <- system.file("extdata", "euratrans",
                    "lv-H3K27me3-BN-male-bio2-tech1.bam",
                     package="chromstaRData")
## Bin the file into bin size 1000bp
data(rn4_chrominfo)
binned <- binReads(file, assembly=rn4_chrominfo, binsizes=1000,
                   stepsizes=500, chromosomes='chr12')
```

```
## Fit the univariate Hidden Markov Model
hmm <- callPeaksUnivariate(binned, max.time=60, eps=1)
temp.file <- tempfile()
save(hmm, file=temp.file)
loaded.hmm <- loadHmmsFromFiles(temp.file)[[1]]
class(loaded.hmm)
```

`mergeChroms` *Merge several* `multiHMM`*s into one object*

Description

Merge several `multiHMM`s into one object. This can be done to merge fits for separate chromosomes into one object for easier handling. Merging will only be done if all models have the same IDs.

Usage

```
mergeChroms(multi.hmm.list, filename = NULL)
```

Arguments

`multi.hmm.list`	A list of `multiHMM` objects or a character vector of files that contain such objects.
`filename`	The file name where the merged object will be stored. If `filename` is not specified, a `multiHMM` is returned.

Value

A `multiHMM` object or NULL, depending on option `filename`.

Author(s)

Aaron Taudt

`model.combined` *Combined multivariate HMM for demonstration purposes*

Description

A `combinedMultiHMM` object for demonstration purposes in examples of package **chromstaR**.

Format

A `combinedMultiHMM` object.

Examples

```
## Get an example combinedMultiHMM
file <- system.file("data","combined_mode-differential.RData",
                    package="chromstaR")
model <- get(load(file))
```

---

model.multivariate *Multivariate HMM for demonstration purposes*

Description

A multiHMM object for demonstration purposes in examples of package **chromstaR**.

Format

A multiHMM object.

Examples

```
## Get an example multiHMM
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
```

---

model.univariate *Univariate HMM for demonstration purposes*

Description

A uniHMM object for demonstration purposes in examples of package **chromstaR**.

Format

A uniHMM object.

Examples

```
## Get an example uniHMM
file <- system.file("data","H3K27me3-BN-rep1.RData", package="chromstaR")
model <- get(load(file))
```

multiHMM *Multivariate HMM object*

Description

The multivariate HMM object is output of the function callPeaksMultivariate and is a list() with various entries. The class() attribute of this list was set to "multiHMM". For a given hmm, the entries can be accessed with the list operators 'hmm[[]]' or 'hmm$'.

Value

A list() with the following entries:

- info — Experiment table for this object.
- bincounts — A GRanges-class object containing the genomic bin coordinates and original binned read count values for different offsets.
- bins — A GRanges-class object containing the genomic bin coordinates, their read count, (optional) posteriors and state classification.
- segments — Same as bins, but consecutive bins with the same state are collapsed into segments.
- peaks — A list() with GRanges-class containing peak coordinates for each ID in info.
- mapping — A named vector giving the mapping from decimal combinatorial states to human readable combinations.
- weights — Weight for each component. Same as apply(hmm$posteriors,2,mean).
- weights.univariate — Weights of the univariate HMMs.
- transitionProbs — Matrix of transition probabilities from each state (row) into each state (column).
- transitionProbs.initial — Initial transitionProbs at the beginning of the Baum-Welch.
- startProbs — Probabilities for the first bin. Same as hmm$posteriors[1,].
- startProbs.initial — Initial startProbs at the beginning of the Baum-Welch.
- distributions — Emission distributions used for this model.
- convergenceInfo — Contains information about the convergence of the Baum-Welch algorithm.
- convergenceInfo$eps — Convergence threshold for the Baum-Welch.
- convergenceInfo$loglik — Final loglikelihood after the last iteration.
- convergenceInfo$loglik.delta — Change in loglikelihood after the last iteration (should be smaller than eps)

convergenceInfo$num.iterations
: Number of iterations that the Baum-Welch needed to converge to the desired eps.

convergenceInfo$time.sec
: Time in seconds that the Baum-Welch needed to converge to the desired eps.

correlation.matrix
: Correlation matrix of transformed reads.

See Also

callPeaksMultivariate, uniHMM, combinedMultiHMM

Examples

```
## Get an example multiHMM
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
```

---

multivariateSegmentation
Multivariate segmentation

Description

Make segmentation from bins for a multiHMM object.

Usage

```
multivariateSegmentation(bins, column2collapseBy = "state")
```

Arguments

bins
: A GRanges-class with binned read counts.

column2collapseBy
: The number of the column which will be used to collapse all other inputs. If a set of consecutive bins has the same value in this column, they will be aggregated into one bin with adjusted genomic coordinates.

Value

A GRanges-class with segmented regions.

plotExpression *Overlap with expression data*

Description

Get the expression values that overlap with each combinatorial state.

Usage

```
plotExpression(hmm, expression, combinations = NULL, return.marks = FALSE)
```

Arguments

hmm	A multiHMM or combinedMultiHMM object or file that contains such an object.
expression	A GRanges-class object with metadata column 'expression', containing the expression value for each range.
combinations	A vector with combinations for which the expression overlap will be calculated. If NULL all combinations will be considered.
return.marks	Set to TRUE if expression values for marks instead of combinations should be returned.

Value

A ggplot2 object if a multiHMM was given or a named list with ggplot2 objects if a combinedMultiHMM was given.

Author(s)

Aaron Taudt

See Also

plotting

Examples

```
## Load an example multiHMM
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))

## Obtain expression data
data(expression_lv)
head(expression_lv)

## We need to get coordinates for each of the genes
library(biomaRt)
ensembl <- useEnsembl(biomart='ENSEMBL_MART_ENSEMBL', dataset='rnorvegicus_gene_ensembl')
```

```
genes <- getBM(attributes=c('ensembl_gene_id', 'chromosome_name', 'start_position',
                            'end_position', 'strand', 'external_gene_name',
                            'gene_biotype'),
               mart=ensembl)
expr <- merge(genes, expression_lv, by='ensembl_gene_id')
# Transform to GRanges
expression.SHR <- GRanges(seqnames=paste0('chr',expr$chromosome_name),
                          ranges=IRanges(start=expr$start, end=expr$end),
                          strand=expr$strand, name=expr$external_gene_name,
                          biotype=expr$gene_biotype,
                          expression=expr$expression_SHR)
# We apply an asinh transformation to reduce the effect of outliers
expression.SHR$expression <- asinh(expression.SHR$expression)

## Plot
plotExpression(model, expression.SHR) +
 theme(axis.text.x=element_text(angle=0, hjust=0.5)) +
 ggtitle('Expression of genes overlapping combinatorial states')
plotExpression(model, expression.SHR, return.marks=TRUE) +
 ggtitle('Expression of marks overlapping combinatorial states')
```

plotGenomeBrowser *#' Plot a genome browser view #' #' Plot a simple genome browser view. This is useful for scripted genome browser snapshots. #' #' @param counts A* GRanges-class *object with meta-data column 'counts'. #' @param peaklist A named list() of* GRanges-class *objects containing peak coordinates. #' @param chr,start,end Chromosome, start and end coordinates for the plot. #' @param countcol A character giving the color for the counts. #' @param peakcols A character vector with colors for the peaks in* peaklist*. #' @param style One of* c('peaks', 'density')*. #' @param peakTrackHeight Relative height of the tracks given in* peaklist *compared to the* counts*. #' @return A* ggplot *object. #' @examples #'## Get an example multiHMM ## #'file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData", #' package="chromstaR") #'model <- get(load(file)) #'## Plot genome browser snapshot #'bins <- model$bins #'bins$counts <- model$bins$counts.rpkm[,1] #'plotGenomeBrowser(counts=bins, peaklist=model$peaks, #' chr='chr12', start=1, end=1e6) #' plotGenomeBrowser2 <- function(counts, peaklist=NULL, chr, start, end, countcol='black', peakcols=NULL, style='peaks', peakTrackHeight=5) ## Select ranges to plot ranges2plot <- reduce(counts[counts@seqnames == chr & start(counts) >= start & start(counts) <= end]) ## Counts counts <- subsetByOverlaps(counts, ranges2plot) if (style == 'peaks') df <- data.frame(x=(start(counts)+end(counts))/2, counts=counts$counts) # plot triangles centered at middle of the bin ggplt <- ggplot(df) + geom_area(aes_string(x='x', y='counts')) + theme(panel.grid = element_blank(), panel.background = element_blank(), axis.text.x = element_blank(), axis.title = element_blank(), axis.ticks.x = element_blank(), axis.line = element_blank()) maxcounts <- max(counts$counts) ggplt <- ggplt + scale_y_continuous(breaks=c(0, maxcounts)) else if (style == 'density') df <- data.frame(xmin=start(counts), xmax=end(counts), counts=counts$counts) ggplt <- ggplot(df) + geom_rect(aes_string(xmin='xmin', xmax='xmax', ymin=0, ymax=4, alpha='counts')) + theme(panel.grid = element_blank(), panel.background = element_blank(), axis.text = element_blank(), axis.title = element_blank(), axis.ticks = element_blank(), axis.line = element_blank()) else stop("Unknown value '", style, "' for parameter 'style'. Must be one of c('peaks', 'density').") ## Peaks if (!is.null(peaklist)) if (is.null(peakcols)) peakcols <- getDistinctColors(length(peaklist)) for (i1 in 1:length(peaklist)) p <- peakTrackHeight peaks <- subsetByOverlaps(peaklist[[i1]], ranges2plot) if (length(peaks) > 0) df <- data.frame(start=start(peaks), end=end(peaks), ymin=-p*i1, ymax=-p*i1+0.9*p) ggplt <- ggplt + geom_rect(data=df, mapping=aes_string(xmin='start', xmax='end', ymin='ymin', ymax='ymax'), col=peakcols[i1], fill=peakcols[i1]) trackname <- names(peaklist)[i1] df <- data.frame(x=start(counts)[1], y=-p*i1+0.5*p, label=trackname) ggplt <- ggplt + geom_text(data=df, mapping=aes_string(x='x', y='y', label='label'), vjust=0.5, hjust=0.5, col=peakcols[i1]) return(ggplt) Plot a genome browser view*

Description

Plot a simple genome browser view of chromstaR-objects. This is useful for scripted genome browser snapshots.

Usage

```
plotGenomeBrowser(
  model,
  chr,
  start,
  end,
  style = "peaks",
  peakHeight = 0.2,
  peakColor = "blue",
  same.yaxis = TRUE
)
```

Arguments

model	A uniHMM, multiHMM or combinedMultiHMM object or file that contains such an object.
chr, start, end	Chromosome, start and end coordinates for the plot.
style	One of c('peaks', 'density').
peakHeight	Height of the peak track relative to the count track.
peakColor	Color for the peak track.
same.yaxis	Whether or not the plots for the same mark have the same y-axis.

Value

A list() of ggplot objects.

Examples

```
## Get an example uniHMM ##
file <- system.file("data","H3K27me3-BN-rep1.RData", package="chromstaR")
model <- get(load(file))
plotGenomeBrowser(model, chr='chr12', start=1, end=1e6, style='peaks',
                  peakHeight=0.1)

## Get an example multiHMM ##
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
plotGenomeBrowser(model, chr='chr12', start=1, end=1e6, style='peaks',
                  peakHeight=0.1)

## Get an example combinedMultiHMM ##
file <- system.file("data","combined_mode-differential.RData",
                    package="chromstaR")
```

```
model <- get(load(file))
plotlist <- plotGenomeBrowser(model, chr='chr12', start=1, end=1e6, style='peaks',
                peakHeight=0.1)
```

`plotHistogram` *Histogram of binned read counts with fitted mixture distribution*

Description

Plot a histogram of binned read counts with fitted mixture distributions from a `uniHMM` object.

Usage

```
plotHistogram(
  model,
  state = NULL,
  chromosomes = NULL,
  start = NULL,
  end = NULL,
  linewidth = 1
)
```

Arguments

`model`	A `uniHMM` object or file that contains such an object.
`state`	Plot the histogram only for the specified state. One of `c('unmodified','modified')`.
`chromosomes, start, end`	Plot the histogram only for the specified chromosomes, start and end position.
`linewidth`	Width of the distribution lines.

Value

A `ggplot` object.

See Also

`plotting`

Examples

```
## Get an example BAM file with ChIP-seq reads
file <- system.file("extdata", "euratrans",
                    "lv-H3K27me3-BN-male-bio2-tech1.bam",
                     package="chromstaRData")
## Bin the BED file into bin size 1000bp
data(rn4_chrominfo)
data(experiment_table)
```

```
binned <- binReads(file, experiment.table=experiment_table,
                   assembly=rn4_chrominfo, binsizes=1000,
                   stepsizes=500, chromosomes='chr12')
plotHistogram(binned)
## Fit the univariate Hidden Markov Model
hmm <- callPeaksUnivariate(binned, max.time=60, eps=1)
## Check if the fit is ok
plotHistogram(hmm)
```

`plotHistograms` *Histograms of binned read counts with fitted mixture distribution*

Description

Plot histograms of binned read counts with fitted mixture distributions from a `multiHMM` object.

Usage

```
plotHistograms(model, ...)
```

Arguments

`model`	A `multiHMM` object or file that contains such an object.
`...`	Additional arguments (see `plotHistogram`).

Value

A `ggplot` object.

See Also

`plotting`

`plotting` *chromstaR plotting functions*

Description

This page provides an overview of all **chromstaR** plotting functions.

Details

Plotting functions that work on uniHMM objects:

plotHistogram Read count histogram with fitted mixture distributions.

Plotting functions that work on multiHMM objects:

heatmapCountCorrelation Heatmap of read count correlations.

heatmapTransitionProbs Heatmap of transition probabilities of the Hidden Markov Model.

heatmapCombinations Binary presence/absence pattern of combinatorial states.

plotExpression Boxplot of expression values that overlap combinatorial states.

Plotting functions that work on multiHMM and combinedMultiHMM objects:

heatmapCountCorrelation Heatmap of read count correlations.

plotEnrichCountHeatmap Heatmap of read counts around annotation.

plotEnrichment Enrichment of combinatorial states around annotation.

plotFoldEnrichHeatmap Enrichment of combinatorial states at multiple annotations.

plotExpression Boxplot of expression values that overlap combinatorial states.

Other plotting functions:

heatmapCombinations Binary presence/absence pattern of combinatorial states.

---

print.combinedMultiHMM

Print combinedMultiHMM object

Description

Print combinedMultiHMM object

Usage

```
## S3 method for class 'combinedMultiHMM'
print(x, ...)
```

Arguments

x	An combinedMultiHMM object.
...	Ignored.

Value

An invisible NULL.

print.multiHMM *Print multiHMM object*

Description

Print multiHMM object

Usage

```
## S3 method for class 'multiHMM'
print(x, ...)
```

Arguments

x	An multiHMM object.
...	Ignored.

Value

An invisible NULL.

print.uniHMM *Print uniHMM object*

Description

Print uniHMM object

Usage

```
## S3 method for class 'uniHMM'
print(x, ...)
```

Arguments

x	An uniHMM object.
...	Ignored.

Value

An invisible NULL.

`readBamFileAsGRanges` *Import BAM file into GRanges*

Description

Import aligned reads from a BAM file into a `GRanges-class` object.

Usage

```
readBamFileAsGRanges(
  bamfile,
  bamindex = bamfile,
  chromosomes = NULL,
  pairedEndReads = FALSE,
  remove.duplicate.reads = FALSE,
  min.mapq = 10,
  max.fragment.width = 1000,
  blacklist = NULL,
  what = "mapq"
)
```

Arguments

`bamfile`	A sorted BAM file.
`bamindex`	BAM index file. Can be specified without the .bai ending. If the index file does not exist it will be created and a warning is issued.
`chromosomes`	If only a subset of the chromosomes should be imported, specify them here.
`pairedEndReads`	Set to `TRUE` if you have paired-end reads in your BAM files (not implemented for BED files).
`remove.duplicate.reads`	A logical indicating whether or not duplicate reads should be removed.
`min.mapq`	Minimum mapping quality when importing from BAM files. Set `min.mapq=0` to keep all reads.
`max.fragment.width`	Maximum allowed fragment length. This is to filter out erroneously wrong fragments due to mapping errors of paired end reads.
`blacklist`	A `GRanges-class` or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.
`what`	A character vector of fields that are returned. Uses the `Rsamtools::scanBamWhat` function. See `Rsamtools::ScanBamParam` to see what is available.

Value

A `GRanges-class` object containing the reads.

Examples

```
## Get an example BAM file with ChIP-seq reads
bamfile <- system.file("extdata", "euratrans", "lv-H3K4me3-BN-female-bio1-tech1.bam",
                       package="chromstaRData")
## Read the file into a GRanges object
reads <- readBamFileAsGRanges(bamfile, chromosomes='chr12', pairedEndReads=FALSE,
                      min.mapq=10, remove.duplicate.reads=TRUE)
print(reads)
```

---

readBedFileAsGRanges *Import BED file into GRanges*

Description

Import aligned reads from a BED file into a GRanges-class object.

Usage

```
readBedFileAsGRanges(
  bedfile,
  assembly,
  chromosomes = NULL,
  remove.duplicate.reads = FALSE,
  min.mapq = 10,
  max.fragment.width = 1000,
  blacklist = NULL
)
```

Arguments

bedfile	A file with aligned reads in BED-6 format. The columns have to be c('chromosome','start','end','descripti
assembly	Please see getChromInfoFromUCSC for available assemblies. Only necessary when importing BED files. BAM files are handled automatically. Alternatively a data.frame with columns 'chromosome' and 'length'.
chromosomes	If only a subset of the chromosomes should be imported, specify them here.
remove.duplicate.reads	A logical indicating whether or not duplicate reads should be removed.
min.mapq	Minimum mapping quality when importing from BAM files. Set min.mapq=0 to keep all reads.
max.fragment.width	Maximum allowed fragment length. This is to filter out erroneously wrong fragments.
blacklist	A GRanges-class or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.

Value

A GRanges-class object containing the reads.

Examples

```
## Get an example BED file with single-cell-sequencing reads
bedfile <- system.file("extdata", "liver-H3K4me3-BN-male-bio2-tech1.bed.gz",
                       package="chromstaRData")
## Read the file into a GRanges object
data(rn4_chrominfo)
reads <- readBedFileAsGRanges(bedfile, assembly=rn4_chrominfo, chromosomes='chr12',
                    min.mapq=10, remove.duplicate.reads=TRUE)
print(reads)
```

---

readConfig *Read chromstaR configuration file*

Description

Read a chromstaR configuration file into a list structure. The configuration file has to be specified in INI format. R expressions can be used and will be evaluated.

Usage

```
readConfig(configfile)
```

Arguments

configfile	Path to the configuration file

Value

A list with one entry for each element in configfile.

Author(s)

Aaron Taudt

`readCustomBedFile` *Read bed-file into GRanges*

Description

This is a simple convenience function to read a bed(.gz)-file into a GRanges-class object. The bed-file is expected to have the following fields: `chromosome`, `start`, `end`, `name`, `score`, `strand`.

Usage

```
readCustomBedFile(
  bedfile,
  col.names = c("chromosome", "start", "end", "name", "score", "strand"),
  col.classes = NULL,
  skip = 0,
  chromosome.format = "NCBI",
  sep = ""
)
```

Arguments

`bedfile`	Filename of the bed or bed.gz file.
`col.names`	A character vector giving the names of the columns in the `bedfile`. Must contain at least `c('chromosome','start','end')`.
`col.classes`	A character vector giving the classes of the columns in `bedfile`. Speeds up the import.
`skip`	Number of lines to skip at the beginning.
`chromosome.format`	Desired format of the chromosomes. Either 'NCBI' for (1,2,3 ...) or 'UCSC' for (chr1,chr2,chr3 ...) or `NULL` to keep the original names.
`sep`	Field separator from `read.table`.

Value

A GRanges-class object with the contents of the bed-file.

Author(s)

Aaron Taudt

Examples

```
## Get an example BED file
bedfile <- system.file("extdata", "liver-H3K4me3-BN-male-bio2-tech1.bed.gz",
                       package="chromstaRData")
## Import the file and skip the first 10 lines
data <- readCustomBedFile(bedfile, skip=10)
```

removeCondition *Remove condition from model*

Description

Remove a condition from a combinedMultiHMM object.

Usage

```
removeCondition(model, conditions)
```

Arguments

model	A combinedMultiHMM object or file which contains such an object.
conditions	A character vector with the condition(s) to be removed.

Value

The input combinedMultiHMM object with specified conditions removed.

Examples

```
## Get an example HMM
file <- system.file("data","combined_mode-differential.RData",
                    package="chromstaR")
model <- get(load(file))

## Print available conditions
print(unique(model$info$condition))

## Remove condition SHR
new.model <- removeCondition(model, conditions='SHR')
```

scanBinsizes *Find the best bin size for a given dataset*

Description

Use simulations to find the best bin size among a set of input files. There is no guarantee that the bin size will be the best for your data, since it is only "best" in terms of fewest miscalls for simulated data. However, it can give you a hint what bin size to choose.

Usage

```
scanBinsizes(
  files.binned,
  outputfolder,
  chromosomes = "chr10",
  eps = 0.01,
  max.iter = 100,
  max.time = 300,
  repetitions = 3,
  plot.progress = FALSE
)
```

Arguments

files.binned	A vector with files that contain binned.data in different bin sizes.
outputfolder	Name of the folder where all files will be written to.
chromosomes	A vector of chromosomes to use for the simulation.
eps	Convergence threshold for the Baum-Welch algorithm.
max.iter	The maximum number of iterations for the Baum-Welch algorithm. The default -1 is no limit.
max.time	The maximum running time in seconds for the Baum-Welch algorithm. If this time is reached, the Baum-Welch will terminate after the current iteration finishes. The default -1 is no limit.
repetitions	Number of repetitions for each simulation.
plot.progress	If TRUE, the plot will be updated each time a simulation has finished. If FALSE, the plot will be returned only at the end.

Details

The function first runs callPeaksUnivariate on the given binned.data files. From the estimated parameters it generates simulated data and calls the peaks on this simulated data. Because the data is simulated, the fraction of miscalls can be precisely calculated.

Value

A ggplot object with a bar plot of the number of miscalls dependent on the bin size.

Author(s)

Aaron Taudt

scores *chromstaR scores*

Description

Various scores used in chromstaR.

Usage

```
differentialScoreMax(mat, info, FUN = "-")

differentialScoreSum(mat, info, FUN = "-")
```

Arguments

mat	A matrix with posterior probabilities, read counts or any other matrix with these dimensions. Column names must correspond to the ID entries in info.
info	An experiment.table with additional column 'ID'.
FUN	A function to compute the score with.

Value

A numeric vector.

Functions

- differentialScoreMax: Maximum differential score. Values are between 0 and 1. A value of 1 means that at least one mark is maximally different between conditions.
- differentialScoreSum: Additive differential score. Values are between 0 and N, where N is the number of marks. A value around 1 means that approximately 1 mark is different, a value of 2 means that 2 marks are different etc.

Author(s)

Aaron Taudt

`simulateMultivariate` *Simulate multivariate data*

Description

Simulate known states, read counts and read coordinates using a multivariate Hidden Markov Model.

Usage

```
simulateMultivariate(
  bins,
  transition,
  emissions,
  weights,
  correlationMatrices,
  combstates,
  IDs,
  fragLen = 50
)
```

Arguments

`bins`	A `GRanges-class` object for which reads will be simulated.
`transition`	A matrix with transition probabilities.
`emissions`	A list() with data.frames with emission distributions (see `uniHMM` entry 'distributions').
`weights`	A list() with weights for the three univariate states.
`correlationMatrices`	A list with correlation matrices.
`combstates`	A vector with combinatorial states.
`IDs`	A character vector with IDs.
`fragLen`	Length of the simulated read fragments.

Value

A `list()` with entries $bins containing the simulated states and read count, $reads with simulated read coordinates.

simulateReadsFromCounts *Simulate read coordinates*

Description

Simulate read coordinates using read counts as input.

Usage

```
simulateReadsFromCounts(bins, fragLen = 50)
```

Arguments

bins	A GRanges-class with read counts.
fragLen	Length of the simulated read fragments.

Value

A GRanges-class with read coordinates.

simulateUnivariate *Simulate univariate data*

Description

Simulate known states, read counts and read coordinates using a univariate Hidden Markov Model with three states ("zero-inflation", "unmodified" and "modified").

Usage

```
simulateUnivariate(bins, transition, emission, fragLen = 50)
```

Arguments

bins	A GRanges-class object for which reads will be simulated.
transition	A matrix with transition probabilities.
emission	A data.frame with emission distributions (see uniHMM entry 'distributions').
fragLen	Length of the simulated read fragments.

Value

A list with entries $bins containing the simulated states and read count, $reads with simulated read coordinates and $transition and $emission.

`state.brewer` *Obtain combinatorial states from specification*

Description

This function returns all combinatorial (decimal) states that are consistent with a given abstract specification.

Usage

```
state.brewer(
  replicates = NULL,
  differential.states = FALSE,
  min.diff = 1,
  common.states = FALSE,
  conditions = NULL,
  tracks2compare = NULL,
  sep = "+",
  statespec = NULL,
  diffstatespec = NULL,
  exclusive.table = NULL,
  binary.matrix = NULL
)
```

Arguments

`replicates`	A vector specifying the replicate structure. Similar entries will be treated as replicates.
`differential.states`	A logical specifying whether differential states shall be returned.
`min.diff`	The minimum number of differences between conditions.
`common.states`	A logical specifying whether common states shall be returned.
`conditions`	A vector with the same length as `replicates`. Similar entries will be treated as belonging to the same condition. Usually your tissue or cell types or time points.
`tracks2compare`	A vector with the same length as `replicates`. This vector defines the tracks between which conditions are compared. Usually your histone marks.
`sep`	Separator used to separate the tracknames in the combinations. The default '+' should not be changed because it is assumed in follow-up functions.
`statespec`	If this parameter is specified, `replicates` will be ignored. A vector composed of any combination of the following entries: `'0.[]'`, `'1.[]'`, `'x.[]'`, `'r.[]'`, where [] can be any string. • `'0.A'`: sample A is 'unmodified' • `'1.B'`: sample B is 'modified'

- 'x.C': sample C can be both 'unmodified' or 'modified'
- 'r.D': all samples in group D have to be in the same state
- 'r.[]': all samples in group [] have to be in the same state

diffstatespec A vector composed of any combination of the following entries: 'x.[]', 'd.[]', where [] can be any string.

- 'x.A': sample A can be both 'unmodified' or 'modified'
- 'd.B': at least one sample in group B has to be different from the other samples in group A
- 'd[]': at least one sample in group [] has to be different from the other samples in group []

exclusive.table
A data.frame or tab-separated text file with columns 'mark' and 'group'. Histone marks with the same group will be treated as mutually exclusive.

binary.matrix A logical matrix produced by dec2bin. If this is specified, only states specified by the rows of this matrix will be considered. The number of columns must match length(replicates) or length(statespec). Only for advanced use. No error handling for incorrect input.

Details

The binary modification state (unmodified=0 or modified=1) of multiple ChIP-seq samples defines a (decimal) combinatorial state such as:

	sample1	sample2	sample3	sample4	sample5	combinatorial state
bin1	0	0	1	0	0	4
bin2	0	0	0	0	0	0
bin3	0	1	0	1	0	10
bin4	0	1	1	1	1	15
bin5	0	0	1	0	1	5

Value

A data.frame with combinations and their corresponding (decimal) combinatorial states.

Author(s)

Aaron Taudt, David Widmann

Examples

```
# Get all combinatorial states where sample1=0, sample2=1, sample3=(0 or 1),
#  sample4=sample5
chromstaR:::state.brewer(statespec=c('0.A','1.B','x.C','r.D','r.D'))

# Get all combinatorial states where sample1=sample2=sample3, sample4=sample5
chromstaR:::state.brewer(statespec=c('r.A','r.A','r.A','r.B','r.B'))
```

```
# Get all combinatorial states where sample1=sample5, sample2=sample3=1,
#  sample4=(0 or 1)
chromstaR:::state.brewer(statespec=c('r.A','1.B','1.C','x.D','r.A'))
```

`stateBrewer` *Obtain combinatorial states from experiment table*

Description

This function computes combinatorial states from an `experiment.table`.

Usage

```
stateBrewer(
  experiment.table,
  mode,
  differential.states = FALSE,
  common.states = FALSE,
  exclusive.table = NULL,
  binary.matrix = NULL
)
```

Arguments

`experiment.table`	A `data.frame` specifying the experiment structure. See `experiment.table`.
`mode`	Mode of brewing. See `Chromstar` for a description of the parameter.
`differential.states`	A logical specifying whether differential states shall be returned.
`common.states`	A logical specifying whether common states shall be returned.
`exclusive.table`	A `data.frame` or tab-separated text file with columns 'mark' and 'group'. Histone marks with the same group will be treated as mutually exclusive.
`binary.matrix`	A logical matrix produced by `dec2bin`. If this is specified, only states specified by the rows of this matrix will be considered. The number of columns must match `length(replicates)` or `length(statespec)`. Only for advanced use. No error handling for incorrect input.

Details

The binary modification state (unmodified=0 or modified=1) of multiple ChIP-seq samples defines a (decimal) combinatorial state such as:

	sample1	sample2	sample3	sample4	sample5	combinatorial state
bin1	0	0	1	0	0	4

bin2	0	0	0	0	0	0
bin3	0	1	0	1	0	10
bin4	0	1	1	1	1	15
bin5	0	0	1	0	1	5

Value

A data.frame with combinations and their corresponding (decimal) combinatorial states.

Author(s)

Aaron Taudt

Examples

```
## Construct an experiment table
data(experiment_table)
print(experiment_table)
## Construct combinatorial states
stateBrewer(experiment_table, mode='combinatorial')
stateBrewer(experiment_table, mode='differential')
stateBrewer(experiment_table, mode='full', common.states=TRUE)

## Exclude states with exclusive.table
excl <- data.frame(mark=c('H3K4me3','H3K27me3'),
                           group=c(1,1))
stateBrewer(experiment_table, mode='full', exclusive.table=excl)
```

---

subsample *Normalize read counts*

Description

Normalize read counts to a given read depth. Reads counts are randomly removed from the input to match the specified read depth.

Usage

```
subsample(binned.data, sample.reads)
```

Arguments

binned.data	A GRanges-class object with meta data column 'reads' that contains the read count.
sample.reads	The number of reads that will be retained.

Value

A GRanges-class object with downsampled read counts.

Author(s)

Aaron Taudt

---

transitionFrequencies *Transition frequencies of combinatorial states*

Description

Get a table of transition frequencies between combinatorial states of different multiHMMs.

Usage

```
transitionFrequencies(
  multi.hmms = NULL,
  combined.hmm = NULL,
  zero.states = "[]",
  combstates = NULL
)
```

Arguments

multi.hmms	A named list with multiHMM objects or a vector with filenames that contain such objects.
combined.hmm	A combinedMultiHMM object. If specified, multi.hmms is ignored.
zero.states	The string(s) which identifies the zero.states.
combstates	Alternative input instead of multi.hmms: A named list of combinatorial state vectors instead of HMMs. Names must be of the form "combination.X", where X is an arbitrary string. If this is specified, multi.hmms and combined.hmm will be ignored.

Value

A data.frame with transition frequencies.

Author(s)

Aaron Taudt

Examples

```
## Get an example combinedMultiHMM
file <- system.file("data","combined_mode-differential.RData",
                    package="chromstaR")
model <- get(load(file))
freqs <- transitionFrequencies(combined.hmm=model)
freqs$table
```

---

uniHMM *Univariate HMM object*

Description

The univariate HMM object is output of the function `callPeaksUnivariate` and is a `list()` with various entries. The `class()` attribute of this list was set to "uniHMM". For a given hmm, the entries can be accessed with the list operators 'hmm[[]]' or 'hmm$'.

Value

A `list()` with the following entries:

`info` Experiment table for this object.

`bincounts` A `GRanges-class` object containing the genomic bin coordinates and original binned read count values for different offsets.

`bins` A `GRanges-class` object containing the genomic bin coordinates, their read count, (optional) posteriors and state classification.

`peaks` A `list()` with `GRanges-class` containing peak coordinates for each ID in `info`.

`weights` Weight for each component. Same as `apply(hmm$posteriors,2,mean)`.

`transitionProbs`
Matrix of transition probabilities from each state (row) into each state (column).

`transitionProbs.initial`
Initial `transitionProbs` at the beginning of the Baum-Welch.

`startProbs` Probabilities for the first bin. Same as `hmm$posteriors[1,]`.

`startProbs.initial`
Initial `startProbs` at the beginning of the Baum-Welch.

`distributions` Estimated parameters of the emission distributions.

`distributions.initial`
Distribution parameters at the beginning of the Baum-Welch.

`post.cutoff` Cutoff for posterior probabilities to call peaks.

`convergenceInfo`
Contains information about the convergence of the Baum-Welch algorithm.

`convergenceInfo$eps`
: Convergence threshold for the Baum-Welch.

`convergenceInfo$loglik`
: Final loglikelihood after the last iteration.

`convergenceInfo$loglik.delta`
: Change in loglikelihood after the last iteration (should be smaller than `eps`)

`convergenceInfo$num.iterations`
: Number of iterations that the Baum-Welch needed to converge to the desired `eps`.

`convergenceInfo$time.sec`
: Time in seconds that the Baum-Welch needed to converge to the desired `eps`.

`convergenceInfo$max.mean`
: Value of parameter `max.mean`.

`convergenceInfo$read.cutoff`
: Cutoff value for read counts.

See Also

`callPeaksUnivariate`, `multiHMM`, `combinedMultiHMM`

`unis2pseudomulti` *Combine univariate HMMs to a multivariate HMM*

Description

Combine multiple `uniHMM`s to a `multiHMM` without running `callPeaksMultivariate`. This should only be done for comparison purposes.

Usage

```
unis2pseudomulti(hmms)
```

Arguments

`hmms`	A named list of `uniHMM` objects. Names will be used to generate the combinations.

Details

Use this function if you want to combine ChIP-seq samples without actually running a multivariate Hidden Markov Model. The resulting object will be of class `multiHMM` but will not be truly multivariate.

Value

A `multiHMM` object.

Author(s)

Aaron Taudt

Examples

```
# Get example BAM files for 2 different marks in hypertensive rat (SHR)
file.path <- system.file("extdata","euratrans", package='chromstaRData')
files <- list.files(file.path, full.names=TRUE, pattern='SHR.*bam$')[c(1,4)]
# Bin the data
data(rn4_chrominfo)
binned.data <- list()
for (file in files) {
 binned.data[[basename(file)]] <- binReads(file, binsizes=1000, stepsizes=500,
                                           assembly=rn4_chrominfo, chromosomes='chr12')
}
# Obtain the univariate fits
models <- list()
for (i1 in 1:length(binned.data)) {
 models[[i1]] <- callPeaksUnivariate(binned.data[[i1]], max.time=60, eps=1)
}
## Combine the univariate HMMs without fitting a multivariate HMM
names(models) <- c('H3K27me3','H3K4me3')
pseudo.multi.HMM <- unis2pseudomulti(models)
## Compare frequencies with real multivariate HMM
exp <- data.frame(file=files, mark=c("H3K27me3","H3K4me3"),
                  condition=rep("SHR",2), replicate=c(1,1), pairedEndReads=FALSE,
                  controlFiles=NA)
states <- stateBrewer(exp, mode='combinatorial')
real.multi.HMM <- callPeaksMultivariate(models, use.states=states, eps=1, max.time=60)
genomicFrequencies(real.multi.HMM)
genomicFrequencies(pseudo.multi.HMM)
```

`variableWidthBins` *Make variable-width bins*

Description

Make variable-width bins based on a reference BAM file. This can be a simulated file (produced by TODO: insert link and aligned with your favourite aligner) or a real reference.

Usage

```
variableWidthBins(reads, binsizes, chromosomes = NULL)
```

Arguments

`reads`	A `GRanges-class` with reads. See `readBamFileAsGRanges` and `readBedFileAsGRanges`.
`binsizes`	A vector with binsizes. Resulting bins will be close to the specified binsizes.
`chromosomes`	A subset of chromosomes for which the bins are generated.

Details

Variable-width bins are produced by first binning the reference BAM file with fixed-width bins and selecting the desired number of reads per bin as the (non-zero) maximum of the histogram. A new set of bins is then generated such that every bin contains the desired number of reads.

Value

A `list()` of GRanges-class objects with variable-width bins.

Author(s)

Aaron Taudt

Examples

```
## Get an example BAM file with ChIP-seq reads
bamfile <- system.file("extdata", "euratrans", "lv-H3K4me3-BN-female-bio1-tech1.bam",
                       package="chromstaRData")
## Read the file into a GRanges object
reads <- readBamFileAsGRanges(bamfile, chromosomes='chr12', pairedEndReads=FALSE,
                    min.mapq=10, remove.duplicate.reads=TRUE)
## Make variable-width bins of size 1000bp
bins <- variableWidthBins(reads, binsizes=1000)
## Plot the distribution of binsizes
hist(width(bins[['1000']]), breaks=50)
```

---

writeConfig *Write chromstaR configuration file*

Description

Write a chromstaR configuration file from a list structure.

Usage

```
writeConfig(conf, configfile)
```

Arguments

`conf`	A list structure with parameter values. Each entry will be written in one line.
`configfile`	Filename of the outputfile.

Value

`NULL`

Author(s)

Aaron Taudt

zinbinom *The Zero-inflated Negative Binomial Distribution*

Description

Density, distribution function, quantile function and random generation for the zero-inflated negative binomial distribution with parameters `w`, `size` and `prob`.

Usage

```
dzinbinom(x, w, size, prob, mu)

pzinbinom(q, w, size, prob, mu, lower.tail = TRUE)

qzinbinom(p, w, size, prob, mu, lower.tail = TRUE)

rzinbinom(n, w, size, prob, mu)
```

Arguments

`x`	Vector of (non-negative integer) quantiles.
`w`	Weight of the zero-inflation. `0 <= w <= 1`.
`size`	Target for number of successful trials, or dispersion parameter (the shape parameter of the gamma mixing distribution). Must be strictly positive, need not be integer.
`prob`	Probability of success in each trial. `0 < prob <= 1`.
`mu`	Alternative parametrization via mean: see 'Details'.
`q`	Vector of quantiles.
`lower.tail`	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
`p`	Vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.

Details

The zero-inflated negative binomial distribution with `size` $= n$ and `prob` $= p$ has density

$$p(x) = w + (1 - w)\frac{\Gamma(x + n)}{\Gamma(n)x!}p^n(1 - p)^x$$

for $x = 0$, $n > 0$, $0 < p \leq 1$ and $0 \leq w \leq 1$.

$$p(x) = (1 - w)\frac{\Gamma(x + n)}{\Gamma(n)x!}p^n(1 - p)^x$$

for $x = 1, 2, \ldots$, $n > 0$, $0 < p \leq 1$ and $0 \leq w \leq 1$.

Value

dzinbinom gives the density, pzinbinom gives the distribution function, qzinbinom gives the quantile function, and rzinbinom generates random deviates.

Functions

- `dzinbinom`: gives the density
- `pzinbinom`: gives the cumulative distribution function
- `qzinbinom`: gives the quantile function
- `rzinbinom`: random number generation

Author(s)

Matthias Heinig, Aaron Taudt

See Also

Distributions for standard distributions, including `dbinom` for the binomial, `dnbinom` for the negative binomial, `dpois` for the Poisson and `dgeom` for the geometric distribution, which is a special case of the negative binomial.

Index