Package 'chromstaR'

April 14, 2017

Type Package

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

Version 1.0.0

Date 2014-05-03

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, S4Vectors, GenomeInfoDb, IRanges, reshape2, Rsamtools, GenomicAlignments, bamsignals

Suggests knitr, BiocStyle, testthat, biomaRt, ggbio

License Artistic-2.0

LazyLoad yes

VignetteBuilder knitr

biocViews Software, DifferentialPeakCalling, HiddenMarkovModel, ChIPSeq, MultipleComparison

RoxygenNote 5.0.1

NeedsCompilation yes

R topics documented:

chromstaR-package . . . 3
binned.data . . . 3
binReads . . . 3
callPeaksMultivariate . . . 5
callPeaksReplicates . . . 7
callPeaksUnivariate . . . 8
callPeaksUnivariateAllChr . . . 11
changePostCutoff . . . 13
Chromstar . . . 14
chromstaR-objects . . . 16

collapseBins . . . 16
combinatorialStates . . . 17
combinedMultiHMM . . . 19
combined_model . . . 19
combineMultivariates . . . 20
conversion . . . 21
enrichmentAtAnnotation . . . 22
enrichment_analysis . . . 23
experiment.table . . . 25
exportBinnedData . . . 26
exportCombinedMultivariate . . . 27
exportGRangesAsBedFile . . . 28
exportMultivariate . . . 29
exportUnivariates . . . 30
fixedWidthBins . . . 31
genomicFrequencies . . . 32
getDistinctColors . . . 33
getStateColors . . . 34
heatmapCombinations . . . 34
heatmapCountCorrelation . . . 35
heatmapTransitionProbs . . . 36
loadHmmsFromFiles . . . 36
mergeChroms . . . 37
multiHMM . . . 38
multivariateSegmentation . . . 39
multivariate_model . . . 39
plotExpression . . . 40
plotHistogram . . . 41
plotHistograms . . . 42
plotKaryogram . . . 42
plotting . . . 43
readBamFileAsGRanges . . . 43
readBedFileAsGRanges . . . 44
readConfig . . . 45
readCustomBedFile . . . 46
scanBinsizes . . . 47
scores . . . 48
simulateReads . . . 48
simulateUnivariate . . . 49
state.brewer . . . 49
stateBrewer . . . 51
subsample . . . 53
transitionFrequencies . . . 54
uniHMM . . . 55
unis2pseudomulti . . . 56
variableWidthBins . . . 57
writeConfig . . . 58
zinbinom . . . 59

Index **61**

`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` 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, assembly, bamindex = file,
  chromosomes = NULL, pairedEndReads = FALSE, min.mapq = 10,
  remove.duplicate.reads = TRUE, max.fragment.width = 1000,
  blacklist = NULL, binsizes = 1000, reads.per.bin = NULL, bins = NULL,
  variable.width.reference = NULL, use.bamsignals = TRUE)
```

Arguments

file	A file with aligned reads. Alternatively a GRanges with aligned reads if format is set to 'GRanges'.
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).
assembly	Please see fetchExtendedChromInfoFromUCSC 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 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.
reads.per.bin	Approximate number of desired reads per bin. The bin size will be selected accordingly.
bins	A named list with GRanges containing precalculated bins produced by fixedWidthBins or variableWidthBins. Names must correspond to 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.

Details

Convert aligned reads from .bam or .bed(.gz) files into read counts in equidistant windows (bins). This function uses countOverlaps to calculate the read counts, or alternatively 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 object with meta data column 'counts' that contains the read count. If multiple binsizes were specified , the function returns a named list() of GRanges 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,
                   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, post.cutoff = NULL,
  keep.posteriors = TRUE, num.threads = 1, max.time = NULL,
  max.iter = NULL, keep.densities = FALSE, verbosity = 1)
```

Arguments

`hmms`	A list of `uniHMM`s 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.
`post.cutoff`	False discovery rate. The default `NULL` means that the state with maximum posterior probability will be chosen, irrespective of its absolute probability.
`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.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.

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(1:2,6:7)]
# 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, 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 `uniHMM`s 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 `uniHMM`s 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, 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, input.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 object with binned read counts or a file that contains such an object.
input.data	Input control for the experiment. A GRanges 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,
                 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, input.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 object with binned read counts or a file that contains such an object.
input.data	Input control for the experiment. A GRanges 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`

changePostCutoff *Change the false discovery rate of a Hidden Markov Model*

Description

Adjusts the peak calls of a uniHMM or multiHMM 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$bins$posterior 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

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,
                 chromosomes='chr12')
## Fit the univariate Hidden Markov Model
# !Keep posteriors to change the post.cutoff later!
hmm <- callPeaksUnivariate(binned, max.time=60, eps=1,
                           keep.posteriors=TRUE)
## Compare fits with different post.cutoffs
plotHistogram(changePostCutoff(hmm, post.cutoff=0.01)) + ylim(0,0.25)
plotHistogram(hmm) + ylim(0,0.25)
plotHistogram(changePostCutoff(hmm, post.cutoff=0.99)) + ylim(0,0.25)
```

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, assembly = NULL, chromosomes = NULL,
  remove.duplicate.reads = TRUE, min.mapq = 10, 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.
assembly	A data.frame or tab-separated file with columns 'chromosome' and 'length'. Alternatively a character specifying the assembly, see fetchExtendedChromInfoFromUCSC 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.
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 $\leq$ 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,
```

```
                                                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 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:

`bins` A `GRanges` 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.separate` A `list` with segments for each condition separately.

See Also

`combineMultivariates`, `uniHMM`, `multiHMM`

`combined_model` *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))
```

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, 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, 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` 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 = c("combinations", "peaks",
  "counts"), num.intervals = 21)
```

Arguments

`bins`	The `$bins` entry from a `multiHMM` or `combinedMultiHMM` object.
`info`	The `$info` entry from a `multiHMM` or `combinedMultiHMM` object.
`annotation`	A `GRanges` 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 which statistic to calculate.
num.intervals	Number of intervals for enrichment 'inside' of 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 object.

Usage

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

plotEnrichCountHeatmap(hmm, annotation, bp.around.annotation = 10000,
  max.rows = 1000)

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

Arguments

hmm	A combinedMultiHMM or multiHMM object or a file that contains such an object.
annotations	A list() with GRanges 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') specifying which statistic to calculate.
combinations	A vector with combinations for which the fold enrichment will be calculated. 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.

annotation — A GRanges 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.

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.

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.
- plotFoldEnrichment: 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 <- useMart('ENSEMBL_MART_ENSEMBL', host='may2012.archive.ensembl.org',
                   dataset='rnorvegicus_gene_ensembl')
genes <- getBM(attributes=c('ensembl_gene_id', 'chromosome_name', 'start_position',
                            'end_position', 'strand', 'external_gene_id',
                            '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$external_gene_id, biotype=genes$gene_biotype)
print(genes)

### Make the enrichment plots ###
# We expect promoter [H3K4me3] and bivalent-promoter signatures [H3K4me3+H3K27me3]
# to be enriched at transcription start sites.
   plotFoldEnrichment(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.
   plotFoldEnrichment(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
   plotFoldEnrichment(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].
   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' and 'pairedEndReads'. Avoid the use of special characters like '-' or '+' as this will confuse the internal file management.

Examples

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

`exportBinnedData` *Export genome browser viewable files*

Description

Export read counts as genome browser viewable file

Usage

```
exportBinnedData(binned.data.list, filename, header = TRUE,
  separate.files = TRUE)
```

Arguments

`binned.data.list`	A `list()` of `binned.data` objects or vector of files that contain such objects.
`filename`	The name of the file that will be written. The ending ".wig.gz" for read counts will be appended. 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 object in `binned.data.list`.

Details

Export read counts from `binned.data` as a file which can be uploaded into a genome browser. Read counts are exported in WIGGLE format (.wig.gz).

Value

`NULL`

Author(s)

Aaron Taudt

See Also

`exportUnivariates`, `exportMultivariate`

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,
                  chromosomes='chr12')
## Export the binned read counts
exportBinnedData(list(binned), filename=tempfile())
```

exportCombinedMultivariate *Export genome browser viewable files*

Description

Export multivariate calls as genome browser viewable file

Usage

```
exportCombinedMultivariate(hmm, filename, what = c("combinations", "peaks"),
  exclude.states = "[]", include.states = NULL, trackname = NULL,
  header = TRUE, separate.files = TRUE)
```

Arguments

hmm	A combinedMultiHMM object or file that contains such an object.
filename	The name of the file that will be written. The ending ".bed.gz" for combinatorial states will be appended. Any existing file will be overwritten.
what	A character vector specifying what will be exported. Supported are c('combinations','peaks','c
exclude.states	A vector of combinatorial states that will be excluded from export.
include.states	A vector of combinatorial states that will be exported. If specified, exclude.states is ignored.
trackname	Name that will be used in the "track name" field of the BED file.
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 condition.

Details

Export combinedMultiHMM objects as files which can be uploaded into a genome browser. Combinatorial states are exported in BED format (.bed.gz).

Value

NULL

Author(s)

Aaron Taudt

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
exportCombinedMultivariate(model, filename=tempfile(), what=c('peaks','combinations'))
```

exportGRangesAsBedFile *Export genome browser viewable files*

Description

Export GRanges as genome browser viewable file

Usage

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

Arguments

gr	A GRanges 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.
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 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

exportUnivariates, exportMultivariate

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,
                 chromosomes='chr12')
plotHistogram(binned)
# Export regions with read count above 20
exportGRangesAsBedFile(binned[binned$counts > 20], filename=tempfile(),
             trackname='read counts above 20')
```

`exportMultivariate` *Export genome browser viewable files*

Description

Export multivariate calls and read counts as genome browser viewable file

Usage

```
exportMultivariate(hmm, filename, what = c("combinations", "peaks", "counts"),
  exclude.states = "[]", include.states = NULL, trackname = NULL,
  header = TRUE, separate.files = TRUE)
```

Arguments

`hmm`	A `multiHMM` object or file that contains such an object.
`filename`	The name of the file that will be written. The appropriate ending will be appended, either ".bed.gz" for combinatorial states and peak-calls or ".wig.gz" for read counts. Any existing file will be overwritten.
`what`	A character vector specifying what will be exported. Supported are `c('combinations', 'peaks', '`
`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.
`trackname`	Name that will be used in the "track name" field of the BED file.
`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 peaks if `what` contains 'peaks' or 'counts'.

Details

Export `uniHMM` objects as files which can be uploaded into a genome browser. Combinatorial states and peak-calls are exported in BED format (.bed.gz) and read counts are exported in WIGGLE format (.wig.gz).

Value

NULL

Author(s)

Aaron Taudt

See Also

exportUnivariates, exportBinnedData

Examples

```
## Get an example multiHMM
file <- system.file("data","multivariate_mode-combinatorial_condition-SHR.RData",
                    package="chromstaR")
model <- get(load(file))
## Export peak calls and combinatorial states
exportMultivariate(model, filename=tempfile(), what=c('peaks','combinations'))
```

---

exportUnivariates *Export genome browser viewable files*

Description

Export univariate peak-calls and read counts as genome browser viewable file

Usage

```
exportUnivariates(hmm.list, filename, what = c("peaks", "counts"),
  header = TRUE, separate.files = TRUE)
```

Arguments

hmm.list	A list() of uniHMM objects or vector of files that contain such objects.
filename	The name of the file that will be written. The appropriate ending will be appended, either ".bed.gz" for peak-calls or ".wig.gz" for read counts. Any existing file will be overwritten.
what	A character vector specifying what will be exported. Supported are c('peaks', 'counts').
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 hmm in hmm.list.

Details

Export uniHMM objects as files which can be uploaded into a genome browser. Peak-calls are exported in BED format (.bed.gz) and read counts are exported in WIGGLE format (.wig.gz).

Value

NULL

Author(s)

Aaron Taudt

See Also

exportBinnedData, exportMultivariate

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,
                   chromosomes='chr12')
## Fit the univariate Hidden Markov Model
hmm <- callPeaksUnivariate(binned, max.time=60, eps=1)
## Export
exportUnivariates(list(hmm), filename=tempfile(), what=c('peaks','counts'))
```

---

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 fetchExtendedChromInfoFromUCSC for available assemblies. This option is ignored if bamfile is specified. Alternatively a data.frame generated by fetchExtendedChromInfoFr
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 objects with fixed-width bins.

Author(s)

Aaron Taudt

Examples

```
## Make fixed-width bins of size 500kb and 1Mb
bins <- fixedWidthBins(assembly='mm10', chromosome.format='NCBI', binsizes=c(5e5,1e6))
bins
```

---

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)
```

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"), exclude.rgb.above = 210)
```

Arguments

n	Number of colors to select.
start.color	Color to start the selection process from.
exclude.colors	Character vector with colors that should not be used.
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

heatmapTransitionProbs *Heatmap of transition probabilities*

Description

Plot a heatmap of transition probabilities for a multiHMM model.

Usage

```
heatmapTransitionProbs(model)
```

Arguments

model	A multiHMM object or file that contains such an object.

Value

A ggplot object.

See Also

plotting

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,
                  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.
`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

`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:

`IDs`	IDs of the input univariate HMMs.
`bins`	A `GRanges` 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.
`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.
`post.cutoff`	False discovery rate. NULL means that the state with maximum posterior probability was chosen, irrespective of its absolute probability (default=NULL).
`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 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 with segmented regions.

---

multivariate_model *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))
```

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 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 named list with expression values.

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 <- useMart('ENSEMBL_MART_ENSEMBL', host='may2012.archive.ensembl.org',
                   dataset='rnorvegicus_gene_ensembl')
genes <- getBM(attributes=c('ensembl_gene_id', 'chromosome_name', 'start_position',
                            'end_position', 'strand', 'external_gene_id',
                            '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$external_gene_id,
                          biotype=expr$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')
```

`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`

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

plotKaryogram — *Plot a karyogram with read counts and univariate peak calls*

Description

Plot a karyogram with read counts and peak calls from a uniHMM object.

Usage

```
plotKaryogram(model)
```

Arguments

model	A uniHMM object or file that contains such an object.

Value

A ggplot object.

Author(s)

Aaron Taudt

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.

`plotKaryogram` Karyogram with read counts and peak calls.

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.

`plotFoldEnrichment` 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.

readBamFileAsGRanges *Import BAM file into GRanges*

Description

Import aligned reads from a BAM file into a `GRanges` 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 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. Type scanBamWhat to see what is available.

Value

A GRanges 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 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','desc
assembly	Please see fetchExtendedChromInfoFromUCSC 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 or a bed(.gz) file with blacklisted regions. Reads falling into those regions will be discarded.

Value

A GRanges 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 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")
```

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 ...).

Value

A GRanges 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)
```

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 or read counts. 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

---

simulateReads *Simulate read coordinates*

Description

Simulate read coordinates using read counts as input.

Usage

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

Arguments

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

Value

A GRanges with read coordinates.

---

simulateUnivariate *Simulate 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 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 object with meta data column 'reads' that contains the read count.
sample.reads	The number of reads that will be retained.

Value

A GRanges 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 `multiHMM`s.

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

```
#=== Step 1: Preparation ===
## Prepare the file paths. Exchange this with your input and output directories.
inputfolder <- system.file("extdata","euratrans", package="chromstaRData")
outputfolder <- file.path(tempdir(), 'SHR-BN-example')
## Define experiment structure
data(experiment_table)
print(experiment_table)
## 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)
head(rn4_chrominfo)

#=== Step 2: Run Chromstar ===
## Run ChromstaR
Chromstar(inputfolder, experiment.table=experiment_table,
```

```
        outputfolder=outputfolder, numCPU=2, binsize=1000, assembly=rn4_chrominfo,
      prefit.on.chr='chr12', chromosomes='chr12', mode='combinatorial', eps.univariate=1,
        eps.multivariate=1)
## Results are stored in 'outputfolder' and can be loaded for further processing
list.files(outputfolder)
model <- get(load(file.path(outputfolder,'combined', 'combined_mode-combinatorial.RData')))

#=== Step 3: Analysis ===
# Get frequencies
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:

`ID`	An identifier that is used in various **chromstaR** functions.
`bins`	A `GRanges` 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.
`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 (default=0.5).
`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(uni.hmm.list)
```

Arguments

`uni.hmm.list`	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,
                                           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 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 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

bamProfile, *4*
bamsignals, *4*
bin2dec, *18*
bin2dec *(*conversion*)*, 21
binned.data, 3, *26*, *47*
binning, *14*
binning *(*binReads*)*, 3
binReads, *3*, 3

callPeaksMultivariate, 5, *8*, *10*, *12*, *16*, *38*, *39*, *56*
callPeaksReplicates, *6*, 7, *16*
callPeaksUnivariate, *5–8*, 8, *16*, *18*, *47*, *55*, *56*
callPeaksUnivariateAllChr, *10*, 11
changePostCutoff, 13
Chromstar, *3*, 14, *20*, *51*
chromstaR, *14–16*, *19*, *22*, *34*, *36*, *39*, *43*, *48*, *55*
chromstaR *(*chromstaR-package*)*, 3
chromstaR-objects, 16
chromstaR-package, 3
collapseBins, 16
colors, *33*
combinatorialStates, 17
combined_model, 19
combinedHMM *(*combinedMultiHMM*)*, 19
combinedMultiHMM, *16*, *19*, 19, *20*, *22*, *23*, *27*, *32*, *35*, *39*, *40*, *43*, *54*, *56*
combineMultivariates, *16*, *19*, 20
conversion, 21
cor, *35*
countOverlaps, *4*

dbinom, *60*
dec2bin, *18*, *50*, *51*
dec2bin *(*conversion*)*, 21
dgeom, *60*
differentialScoreMax *(*scores*)*, 48
differentialScoreSum *(*scores*)*, 48
Distributions, *60*
dnbinom, *60*
dpois, *60*
dzinbinom *(*zinbinom*)*, 59

enrichment_analysis, 23
enrichmentAtAnnotation, 22
experiment.table, *4*, *14*, 25, *48*, *51*
exportBinnedData, 26, *30*, *31*
exportCombinedMultivariate, 27
exportGRangesAsBedFile, 28
exportMultivariate, *26*, *28*, 29, *31*
exportUnivariates, *26*, *28*, *30*, 30

fetchExtendedChromInfoFromUCSC, *4*, *14*, *31*, *45*
fixedWidthBins, *4*, 31

genomicFrequencies, 32
getDistinctColors, 33
getStateColors, 34
ggplot, *24*, *34–36*, *41*, *42*, *47*
GRanges, *3*, *4*, *9*, *11*, *19*, *22–24*, *28*, *32*, *38–40*, *43–46*, *49*, *53*, *55*, *57*, *58*

heatmapCombinations, 34, *43*
heatmapCountCorrelation, 35, *43*
heatmapTransitionProbs, 36, *43*

loadHmmsFromFiles, 36

mergeChroms, 37
multi.hmm *(*multiHMM*)*, 38
multiHMM, *6–8*, *13*, *16*, *19*, *20*, *22*, *23*, *29*, *32*, *34–37*, 38, *39*, *40*, *42*, *43*, *54*, *56*
multivariate_model, 39
multivariateSegmentation, 39

plotEnrichCountHeatmap, *43*
plotEnrichCountHeatmap *(*enrichment_analysis*)*, 23
plotExpression, 40, *43*
plotFoldEnrichHeatmap, *43*
plotFoldEnrichHeatmap *(*enrichment_analysis*)*, 23
plotFoldEnrichment, *43*
plotFoldEnrichment *(*enrichment_analysis*)*, 23
plotHistogram, 41, *42*, *43*
plotHistograms, 42

plotKaryogram, 42, *43*
plotting, *3*, *24*, *34–36*, *40–42*, 43
pzinbinom *(*zinbinom*)*, 59

qzinbinom *(*zinbinom*)*, 59

readBamFileAsGRanges, 43, *57*
readBedFileAsGRanges, *14*, 44, *57*
readConfig, 45
readCustomBedFile, 46
rzinbinom *(*zinbinom*)*, 59

scanBamWhat, *44*
scanBinsizes, 47
scores, 48
simulateReads, 48
simulateUnivariate, 49
state.brewer, 49
stateBrewer, *5*, 51
subsample, 53

transitionFrequencies, 54

uni.hmm *(*uniHMM*)*, 55
uniHMM, *5*, *7*, *8*, *10*, *12*, *13*, *16*, *19*, *29*, *30*, *39*, *41–43*, *49*, 55, *56*
unis2pseudomulti, 56

variableWidthBins, *4*, 57

writeConfig, 58

zinbinom, 59