Package 'epigraHMM'

December 20, 2024

Title Epigenomic R-based analysis with hidden Markov models

Version 1.15.0

Date 2019-12-10

biocViews ChIPSeq, ATACSeq, DNaseSeq, HiddenMarkovModel, Epigenetics

Description epigraHMM provides a set of tools for the analysis of epigenomic data based on hidden Markov Models. It contains two separate peak callers, one for consensus peaks from biological or technical replicates, and one for differential peaks from multi-replicate multi-condition experiments. In differential peak calling, epigraHMM provides window-specific posterior probabilities associated with every possible combinatorial pattern of read enrichment across conditions.

License MIT + file LICENSE

Imports Rcpp, magrittr, data.table, SummarizedExperiment, methods, GenomeInfoDb, GenomicRanges, rtracklayer, IRanges, Rsamtools, bamsignals, csaw, S4Vectors, limma, stats, Rhdf5lib, rhdf5, Matrix, MASS, scales, ggpubr, ggplot2, GreyListChIP, pheatmap, grDevices

LinkingTo Rcpp, RcppArmadillo, Rhdf5lib

RoxygenNote 7.2.3

Encoding UTF-8

SystemRequirements GNU make

Suggests testthat, knitr, rmarkdown, BiocStyle, BSgenome.Rnorvegicus.UCSC.rn4, gcapc, chromstaRData

VignetteBuilder knitr

LazyData true

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

git_branch devel

git_last_commit 524cd3d

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2024-12-20

Author Pedro Baldoni [aut, cre]

Maintainer Pedro Baldoni <pedrobaldoni@gmail.com>

Contents

`addOffsets` *Add offsets to epigraHMMDataSet*

Description

This function adds model offsets to epigraHMMDataSet

Usage

```
addOffsets(object, offsets)
```

Arguments

`object`	an epigraHMMDataSet
`offsets`	a matrix with model offsets

Details

To be added

Value

An epigraHMMDataSet with an 'offsets' assay filled in.

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- list('counts' = matrix(rpois(4e5,10),ncol = 4),
'controls' = matrix(rpois(4e5,5),ncol = 4))
colData <- data.frame(condition = c('A','A','B','B'), replicate = c(1,2,1,2))
object <- epigraHMMDataSetFromMatrix(countData,colData)

# Adding pre-computed offsets
object <- addOffsets(object = object,
                     offsets = matrix(rnorm(4e5),ncol = 4))
```

---

callPatterns *Extract posterior probabilities (or combinatorial patterns) associated with differential regions*

Description

Given results from epigraHMM's differential peak caller, this function will output either posterior probabilities or combinatorial patterns associated with the mixture components of the embedded mixture model.

Usage

```
callPatterns(
  object,
  peaks,
  hdf5 = metadata(object)$output,
  type = "all",
  fdr = NULL,
  pattern = NULL,
  ranges = NULL
)
```

Arguments

object	an epigraHMMDataSet
peaks	a GRanges object with differential peaks from 'callPeaks'
hdf5	a character with the location of the epigraHMM HDF5 output file

type	a character string that defines which output will be givem (see details; default is 'all')
fdr	the desired fdr thresholding level to define combinatorial patterns
pattern	a string that explicitly specifies the combinatorial pattern to be output
ranges	a GRanges object with the genomic ranges to subset the output

Details

The output of 'callPatterns' is always restricted to genomic windows intersecting peaks.

If 'type = 'all'', all windows' posterior probabilities associated with all differential combinatorial patterns are returned. If 'type = 'fdr'', users must also specify the input argument 'pattern' and this function will output windows wich are associated with the given 'pattern' that pass a particular fdr threshold level. If 'type = 'max'', this function will output the combinatorial pattern which has the maximal posterior probability for each window. If 'type = 'ranges'', the windows that are output are restricted to those that intersect the 'ranges' input argument.

Value

A GRanges object with metadata

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- cbind(rbind(matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1)),
                   rbind(matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1)))

colData <- data.frame(condition = c('A','B'), replicate = c(1,1))
rowRanges <- GenomicRanges::GRanges('chrA',
                     IRanges::IRanges(start = seq(1,by = 500,
                     length.out = nrow(countData)),width = 500))
```

```
object <- epigraHMMDataSetFromMatrix(countData,colData,rowRanges = rowRanges)

# Initializing
object <- initializer(object,controlEM())

# Running epigraHMM
object <- epigraHMM(object,controlEM(),type = 'differential',dist = 'nb')

# Calling peaks
peaks <- callPeaks(object = object,
                   hdf5 = S4Vectors::metadata(object)$output,
                   method = 'viterbi')

# Extracting posterior probabilities
patterns <- callPatterns(object = object,peaks = peaks,type = 'max')
```

`callPeaks` *Summarize peak calls and optionally create a BED 6+3 file in broadPeak format for visualization*

Description

This function imports the output from 'epigraHMM' and outputs a set of peaks (consensus or differential) for a given FDR control threshold or Viterbi sequence.

Usage

```
callPeaks(
  object,
  hdf5 = metadata(object)$output,
  method = "viterbi",
  saveToFile = FALSE,
  control = NULL
)
```

Arguments

`object`	an epigraHMMDataSet
`hdf5`	a character with the location of the epigraHMM HDF5 output file
`method`	either 'viterbi' or a numeric FDR control threshold (e.g. 0.05). Default is 'viterbi'.
`saveToFile`	a logical indicating whether or not to save the results to file. Output files are always saved with peaks of interest defined on the region level. Default is FALSE.
`control`	list of control arguments from controlEM(). This is an optional parameter and it is only required when 'saveToFile = TRUE' so that the output directory can be obtained. Default is NULL.

Value

A GRanges object with differential peak calls in BED 6+3 format

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- rbind(matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1),
                   matrix(rnbinom(2e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1))

colData <- data.frame(condition = 'A', replicate = 1)

rowRanges <- GenomicRanges::GRanges('chrA',
IRanges::IRanges(start = seq(from = 1, length.out = 4e3,by = 250),width = 250))

object <- epigraHMMDataSetFromMatrix(countData,colData,rowRanges)

# Initializing
object <- initializer(object,controlEM())

# Running epigraHMM
object <- epigraHMM(object,controlEM(),type = 'consensus',dist = 'nb')

# Calling peaks
peaks <- callPeaks(object = object,
                   hdf5 = S4Vectors::metadata(object)$output,
                   method = 'viterbi')
```

---

cleanCounts *Remove effects from covariates of interest*

Description

This function removes the effect from covariates of interest (such as GC content) from experimental counts

Usage

```
cleanCounts(object, effectNames, byNames = NULL, log = TRUE)
```

Arguments

object	an epigraHMMDataSet
effectNames	a character vector with the names of assays for which the effect will be removed from the experimental counts. Names in 'effectNames' must be assays stored in the epigraHMMDataSet 'object'.
byNames	a character vector with the name of an assay containing stratification variables which will be used to define stratum-specific effects. Examples of byNames assays include the 'peaks' assay from 'initializer()'. In this case, models will be fit separately for peaks and non-peaks regions. This can be useful for effects such as GC content, which are known to have a differential effect between peaks and non-peak regions. Default is NULL, i.e., effects will be removed without stratification.
log	a logical indicating if the effect from 'effectNames' should be log-transformed in the regression model (default is TRUE)

Value

An epigraHMMDataSet with an 'offset' assay filled in.

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
gc <- rbeta(3e3,50,50)

countData <- list('counts' = rbind(matrix(rnbinom(2e3,mu = 7.5,size = 10),ncol = 1),
                                   matrix(rnbinom(3e3,mu = exp(0.5 + 8*gc),size = 5),ncol = 1),
                                         matrix(rnbinom(2e3,mu = 7.5,size = 10),ncol = 1)),
                  'gc' = matrix(c(rbeta(2e3,50,50),gc,rbeta(2e3,50,50)),ncol = 1))

colData <- data.frame(condition = 'A', replicate = 1)
object <- epigraHMMDataSetFromMatrix(countData,colData)

# Initializing
object <- initializer(object = object,controlEM())

# Cleaning counts
object <- cleanCounts(object = object,effectNames = 'gc',byNames = 'peaks')

# Plotting the cleaned data
#par(mfrow = c(2,1))
#smoothScatter(log1p(assay(object))~assay(object,'gc'),xlab = 'gc',ylab = 'log counts')
#smoothScatter(as.numeric(log(assay(object)+1) - assay(object,'offsets'))~assay(object,'gc'),
```

```
#              xlab = 'gc',ylab = 'log cleaned counts')
```

---

controlEM *Control parameters for the EM algorithm from epigraHMM*

Description

This function passes controlling parameters for the EM algorithm implemented in the epigraHMM package.

Usage

```
controlEM(
  epsilonEM = c(MRCPE = 0.001, MACPE = 0.001, ARCEL = 0.001),
  maxIterEM = 500,
  minIterEM = 3,
  gapIterEM = 3,
  maxCountEM = 3,
  maxDisp = 1000,
  criterion = "all",
  minZero = .Machine$double.xmin,
  probCut = 0.05,
  quiet = TRUE,
  maxIterInnerEM = 5,
  epsilonInnerEM = 0.001,
  trimOffset = 3,
  pattern = NULL,
  tempDir = tempdir(),
  fileName = "epigraHMM",
  pruningThreshold = NULL,
  quietPruning = TRUE
)
```

Arguments

epsilonEM	a named vector of positive values specifying up to four possible convergence criterion tolerances for the EM algorithm (see 'criterion' below). Default is c('MRCPE' = 1e-3, 'MACPE' = 1e-3,'ARCEL' = 1e-3).
maxIterEM	a positive integer giving the maximum number of EM iterations. Default is 500.
minIterEM	a positive integer giving the minimum number of EM iterations to start evaluating the convergence. Default is 3.
gapIterEM	a positive integer giving the number of EM iterations apart to compute the convergence criterion. Default is 3.
maxCountEM	a positive integer giving the number of consecutive EM iterations satisfying the convergence criterion in order to stop the algorithm. Default is 3.

maxDisp	a positive value for the upper limit constraint of the dispersion parameters. Default is 1000.
criterion	a character specifying the convergence criterion. Either "MRCPE" (maximum absolute relative change in parameter estimates), "MACPE" (maximum absolute change of parameter estimates), "ARCEL" (absolute relative change of the Q-function), or "all" (simultaneously check for MRCPE, MACPE, and ARCEL). Default is "all".
minZero	a positive value for the minimum positive value allowed in computations to avoid having zeros. Default is .Machine$double.xmin.
probCut	a number between 0 and 1 for the cutoff of the rejection controlled EM algorithm. Default 0.05.
quiet	a logical indicating whether to print messages. Default is TRUE.
maxIterInnerEM	a positive integer giving the maximum number of inner EM iterations. Default is 5.
epsilonInnerEM	a positive value with the convergence tolerance value for the inner EM algorithm. The criterion for the inner EM is "MRCPE". Default is 1e-3.
trimOffset	either NULL or a positive integer indicating the number of decimal places to be used in the offset. Default is 3.
pattern	either NULL (the default) or a list with length equal to the number of differential patterns to be modeled by the differential HMM state. See Details section below.
tempDir	a string where results will be saved. Default is 'tempdir()'.
fileName	a string with the name of the result files. Default is 'epigraHMM'.
pruningThreshold	a numeric value between 0 and 1 to consider when pruning rare combinatorial patterns. Default is NULL (see Details).
quietPruning	a logical indicating whether to print messages during the pruning step. Default is TRUE.

Details

If pattern is NULL, every possible combinatorial pattern will be considered. If pattern is a list, elements of it should specify the differential patterns to be modeled by each mixture component. For instance, if pattern = list(2,c(1,3)) the mixture model will have two components that will represent the enrichment of condition 2 alone and the enrichment of conditions 1 and 3 together.

If pruningThreshold is a value between 0 and 1, say 0.05, epigraHMM will sequentially remove differential combinatorial patterns of enrichment from any mixture model component with associated posterior mixture proportion less than 0.05.

Value

A list with components equal to the arguments

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# No more than 100 EM iterations
control <- controlEM(maxIterEM = 100)
```

---

epigraHMM *Perform peak calling of epigenomic data sets*

Description

This function runs either consensus (one condition, multiple samples) or differential (multiple conditions and samples) peak callers for epigenomic data.

Usage

```
epigraHMM(object, control, type, dist = "nb")
```

Arguments

object	an epigraHMMDataSet
control	list of control arguments from controlEM
type	character, either "consensus" or "differential"
dist	character, either "zinb" or "nb" (default)

Value

An epigraHMMDataSet object with the results from epigraHMM

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- rbind(matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1),
                   matrix(rnbinom(2e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1))

colData <- data.frame(condition = 'A', replicate = 1)
object <- epigraHMMDataSetFromMatrix(countData,colData)

# Initializing
object <- initializer(object,controlEM())

# Running epigraHMM
object <- epigraHMM(object,controlEM(),type = 'consensus',dist = 'nb')
```

`epigraHMMDataSetFromBam` *Create a epigraHMMDataSet from a set of BAM files*

Description

This function creates a `RangedSummarizedExperiment` object from of a set of BAM files. It is used to store the input data, the model offsets, and the results from the peak calling algorithms.

Usage

```
epigraHMMDataSetFromBam(
  bamFiles,
  colData,
  genome,
  windowSize,
  gapTrack = TRUE,
  blackList = TRUE
)
```

Arguments

`bamFiles`	a string vector (or a list of string vectors) with the path for BAM files. If bamFiles is a list of string vectors, vectors must be named, have the same dimension, and, at least, a vector with name 'counts' must exist (see details).
`colData`	a `data.frame` with the experimental data. It must contain the columns `condition` and `replicate`. `condition` refers to the experimental condition identifier (e.g. cell line name). `replicate` refers to the replicate identification number (unique for each condition).
`genome`	either a single string with the name of the reference genome (e.g. 'hg19') or a GRanges object with ranges to be tilled into a set of non-overlapping windows.

`windowSize` an integer specifying the size of genomic windows where read counts will be computed.

`gapTrack` either a logical (`TRUE`, the default, or `FALSE`) or a GRanges object with gap regions of the genome to be excluded. If `TRUE`, the function will discard genomic coordinates overlapping regions present in the UCSC gap table of the respective reference genome (if available). See Details section below.

`blackList` either a logical (`TRUE`, the default, or `FALSE`) or a GRanges object with blacklisted regions of the genome to be excluded. If `TRUE`, the function will discard ENCODE blacklisted regions from selected reference genomes (if available). See Details section below.

Details

The index ".bai" files must be stored in the same directory of their respective BAM files. The index files must be named after their respective BAM files with the additional ".bai" suffix.

'epigraHMMDataSetFromBam' will store experimental data (e.g. ChIP-seq counts) from bamFiles (or bamFiles[['counts']], if a list is provided). Additional data (e.g. input control counts) will be stored similarly with their respective list names.

By default, the function computes read counts using csaw's estimated fragment length via cross correlation analysis. For experimental counts (e.g. ChIP-seq), sequencing reads are shifted downstream half of the estimated fragment length. For additional counts (e.g. input control), sequencing reads are not shifted prior to counting.

Additional columns included in the colData input will be passed to the resulting epigraHMMDataSet assay and can be acessed via `colData()` function.

The `genome` argument will call GenomeInfoDb::Seqinfo() to fetch the chromosome lengths of the specified genome. See ?GenomeInfoDb::Seqinfo for the list of UCSC genomes that are currently supported.

If `gapTrack = TRUE` and the name of a reference genome is passed as input through `genome` (e.g. 'hg19'), the function will discard any genomic coordinate overlapping regions specified by the respective UCSC gap table. If `gapTrack` is a GRanges object, the function will discard any genomic coordinate overlaping regions from `gapTrack`.

If `blackList = TRUE` and the name of a reference genome is passed as input through `genome` (e.g. 'hg19'), The function will fetch the manually curated blacklist tracks (Version 2) from https://github.com/Boyle-Lab/Blacklist/tree/master/lists. Current available genomes are ce10, dm3, hg19, hg38, and mm10. If `blackList` is a GRanges object, the function will discard any genomic coordinate overlaping regions from `blackList`.

Value

An epigraHMMDataSet object with sorted colData regarding conditions and replicates. Experimental counts will be stored in the 'counts' assay in the resulting epigraHMMDataSet object. Additional experimental data will be stored with their respective names from the list bamFiles.

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM DOI: 10.1093/nar/gkv1191 DOI: 10.1038/s41598-019-45839-z DOI: 10.1038/nature11247

Examples

```
bamFiles <- system.file("extdata","euratrans",
                        "lv-H3K27me3-SHR-male-bio2-tech1.bam",
                        package="chromstaRData")

colData <- data.frame(condition = 'SHR', replicate = 1)

object <- epigraHMMDataSetFromBam(bamFiles = bamFiles,
                                  colData = colData,
                                  genome = 'rn4',
                                  windowSize = 25000,
                                  gapTrack = TRUE,
                                  blackList = TRUE)
```

`epigraHMMDataSetFromMatrix` *Create a epigraHMMDataSet from matrices of counts*

Description

This function creates a `RangedSummarizedExperiment` object from matrices of counts. It is used to store the input data, the model offsets, and the results from the peak calling algorithms.

Usage

```
epigraHMMDataSetFromMatrix(countData, colData, rowRanges = NULL)
```

Arguments

`countData`	a matrix (or a list of matrices). If countData is a list of matrices, matrices must be named, have the same dimensions, and, at least, a matrix with name 'counts' must exist (see details).
`colData`	a `data.frame` with columns `condition` and `replicate`. `condition` refers to the experimental condition identifier (e.g. cell line name). `replicate` refers to the replicate identification number (unique for each condition).
`rowRanges`	an optional GRanges object with the genomic coordinates of the `countData`

Details

Additional columns included in the colData input will be passed to the resulting epigraHMMDataSet assay and can be acessed via `colData()` function.

Value

An epigraHMMDataSet object with sorted colData regarding conditions and replicates. Experimental counts will be stored in the 'counts' assay in the resulting epigraHMMDataSet object. If 'countData' is a list of matrices, the resulting 'counts' assay will be equal to 'countData[['counts']]'.

Additional matrices can be included in the epigraHMMDataSet. For example, if one wants to include counts from an input control experiment from 'countData[['controls']]', an assay 'control' will be added to the resulting epigraHMMDataSet..

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
countData <- list('counts' = matrix(rpois(4e5,10),ncol = 4),
'controls' = matrix(rpois(4e5,5),ncol = 4))
colData <- data.frame(condition = c('A','A','B','B'), replicate = c(1,2,1,2))
object <- epigraHMMDataSetFromMatrix(countData,colData)
```

---

estimateTransitionProb

Estimate transition probability from a sequence of integers

Description

This function estimates the transition probabilities for a k-state Markov chain based on a sequence of integers that represent states of the chain

Usage

```
estimateTransitionProb(chain, numStates)
```

Arguments

chain	a vector of integers
numStates	an integer, the number of states in the Markov chain

Value

A k-by-k matrix of transition probabilities, such that k is the number of states of the chain

References

https://github.com/plbaldoni/epigraHMM

Examples

```
trueMat <- matrix(c(0.9,0.1,0.1,0.9),2,2)
simChain <- simulateMarkovChain(trueMat,1e3)
estMat <- estimateTransitionProb(simChain,2)

# estMat should be close to trueMat
estMat
```

`expStep` *E-step of HMM (forward-backward probability + posterior probability calculation)*

Description

E-step of HMM (forward-backward probability + posterior probability calculation)

Usage

```
expStep(pi, gamma, logf, hdf5)
```

Arguments

`pi`	a vector of probabilities (sum of probabilities should sum to one)
`gamma`	a matrix of transition probabilities (row sums should be one)
`logf`	a matrix of observed log-likelihood values. Columns represent hidden states, rows represent genomic regions
`hdf5`	path to where the hdf5 is saved

Examples

```
#Creating dummy object
countData <- rbind(matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1),
                   matrix(rnbinom(2e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1))



colData <- data.frame(condition = 'A', replicate = 1)
object <- epigraHMMDataSetFromMatrix(countData,colData)

#Initializing
object <- initializer(object,controlEM())
```

```
#Running epigraHMM
object <- epigraHMM(object,controlEM(),type = 'consensus',dist = 'nb')

#Example
expStep(pi = c(0.99,0.02),
       gamma = matrix(c(0.99,0.01,0.01,0.99),nrow = 2),
       logf = cbind(dnbinom(rnbinom(100,mu = 2,size = 10),mu = 2,size = 10,log = TRUE),
                    dnbinom(rnbinom(100,mu = 7.5,size = 5),mu = 7.5,size = 5,log = TRUE)),
       hdf5 = file.path(tempdir(),'tmp.h5'))
```

`helas3` *ENCODE ChIP-seq broad data from Helas3 cell line*

Description

Data from EZH2, H3K27me3, and H3K36me3 ChIP-seq data from Helas3 cell line. For illustrative purposes, the data has been subset to chromosome 19. The dataset contains two replicates from each mark.

Usage

```
data(helas3)
```

Format

An object of class `"epigraHMMDataSet"`.

Source

ENCODE Broad Histone

References

Davis et al. (2018) NAR 46(D1):D794-D801. (PubMed)

Examples

```
# # # The data 'helas3' was created as follows.
# options(timeout=9999999)
#
# url <- 'http://hgdownload.soe.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeBroadHistone/'
# samples <- c('wgEncodeBroadHistoneHelas3H3k36me3StdAlnRep1.bam',
#              'wgEncodeBroadHistoneHelas3H3k36me3StdAlnRep2.bam',
#              'wgEncodeBroadHistoneHelas3H3k27me3StdAlnRep1.bam',
#              'wgEncodeBroadHistoneHelas3H3k27me3StdAlnRep2.bam',
#              'wgEncodeBroadHistoneHelas3Ezh239875AlnRep1.bam',
#              'wgEncodeBroadHistoneHelas3Ezh239875AlnRep2.bam')
#
# input <- paste0(url,samples)
# output <- paste0(tempdir(),samples)
```

```
#
# for(idx in seq_len(length(input))){
#     download.file(url = input[idx],destfile = output[idx])
#     download.file(url = paste0(input[idx],'.bai'),
#                   destfile = paste0(output[idx],'.bai'))
# }
#
# gr <- segmentGenome(genome = 'hg19',
#                     window = 1000,rm.gap = TRUE,rm.blacklist = TRUE)
#
# cData <- data.frame(condition = rep(c('H3K36me3','H3K27me3','EZH2'),each = 2),
#                     replicate = rep(c(1,2),times = 3))
#
# subGr <- gr[seqnames(gr) == 'chr19' & start(gr) >= 40e6 & end(gr) <= 50e6]
#
# helas3 <-
#     epigraHMMDataSetFromBam(bamFiles = output,colData = cData,
#                             genome = subGr,windowSize = 1000)

data(helas3)
helas3
```

`info` *Get information about peak calling results*

Description

This function returns the BIC and expected log-likelihood function of the model, with respect to the last conditional distribution of unknown enrichment peaks given the data. The latter is also known as 'Q-function' in the EM context.

Usage

```
info(object)
```

Arguments

`object`	an epigraHMMDataSet

Value

A list with BIC, and expected log-likelihood function of the model. If the input object contains results from a differential analysis, 'info' will also output the enrichment patterns associated with each mixture component used in the mixture model.

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- rbind(matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1),
                   matrix(rnbinom(2e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1))

colData <- data.frame(condition = 'A', replicate = 1)
object <- epigraHMMDataSetFromMatrix(countData,colData)

# Initializing
object <- initializer(object,controlEM())

# Running epigraHMM
object <- epigraHMM(object,controlEM(),type = 'consensus',dist = 'nb')

# Get info
info(object)
```

---

initializer *Initializer of epigraHMM*

Description

This function call enriched windows individually for each sample in an epigraHMMDataSet. These are then used for initializing purposes in epigraHMM. By default, the Viterbi algorithm is used to determine enriched windows. Input controls and normalizing offsets are not utilized in this initialization step.

Usage

```
initializer(object, control)
```

Arguments

object	an epigraHMMDataSet
control	list of control arguments from controlEM()

Details

To be added

Value

An epigraHMMDataSet with a 'peaks' assay filled in.

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- rbind(matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1),
                   matrix(rnbinom(2e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1))

colData <- data.frame(condition = 'A', replicate = 1)
object <- epigraHMMDataSetFromMatrix(countData,colData)

# Initializing
object <- initializer(object,controlEM())

# Visualizing initialization peaks
#plot(assay(object),type = 'l')
#lines(7.5*assay(object,'peaks'),col = 'red')
```

---

maxStepProb *M-step (maximization w.r.t. initial and transition probabilities)*

Description

M-step (maximization w.r.t. initial and transition probabilities)

Usage

```
maxStepProb(hdf5)
```

Arguments

hdf5	path to where the hdf5 is saved

Examples

```
#Creating dummy object
countData <- rbind(matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1),
                   matrix(rnbinom(2e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1))
```

```
colData <- data.frame(condition = 'A', replicate = 1)
object <- epigraHMMDataSetFromMatrix(countData,colData)

#Initializing
object <- initializer(object,controlEM())

#Running epigraHMM
object <- epigraHMM(object,controlEM(),type = 'consensus',dist = 'nb')

#Example
maxStepProb(hdf5 = S4Vectors::metadata(object)$output)
```

`normalizeCounts` *Normalize counts*

Description

This function performs a non-linear normalization of counts with respect to a reference sample (geometric mean)

Usage

```
normalizeCounts(object, control, span = 1, ...)
```

Arguments

`object`	an epigraHMMDataSet
`control`	list of control arguments from controlEM()
`span`	the span parameter of `loessFit` (default is 1)
`...`	arguments to be passed to `loessFit` for loess calculation

Details

This function ‘limma::loessFit‘, which simply a wrapper for the ‘stats::lowess‘ smoother.

Value

An epigraHMMDataSet with an ’offsets’ assay filled in.

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- list('counts' = matrix(rpois(1e5,10),ncol = 2),
'controls' = matrix(rpois(1e5,5),ncol = 2))
colData <- data.frame(condition = c('A','A'), replicate = c(1,2))
object <- epigraHMMDataSetFromMatrix(countData,colData)

# Normalizing counts
object <- normalizeCounts(object = object,control = controlEM(), span = 1)
```

---

plotCounts *Create a plot with the results from epigraHMM*

Description

'plotCounts()' plots read counts and peak regions from 'epigraHMM()'

Usage

```
plotCounts(
  object,
  ranges,
  hdf5 = metadata(object)$output,
  peaks = NULL,
  annotation = NULL
)
```

Arguments

object	an epigraHMMDataSet
ranges	a GRanges object or a pair of integers with the genomic corrdinates/windows to be plotted
hdf5	an optional character string with the hdf5 file path from 'epigraHMM'
peaks	an optional parameter with a GRanges object or a vector of logicals (with length equal to the number of rows in 'object') specifying the genomic corrdinates/windows with peaks
annotation	an optional parameter with a GRanges object or a vector of logicals (with length equal to the number of rows in 'object') specifying the genomic corrdinates/windows of an annotation track

Details

If the input object contains the assay 'offset', reads will be normalized prior to plotting (e.g. counts/exp(offset)). Reads from replicates pertaining to the same condition are aggregated prior to plotting.

Value

A ggplot

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
countData <- rbind(matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1),
                   matrix(rnbinom(1e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 7.5,size = 5),ncol = 1),
                   matrix(rnbinom(1e3,mu = 2,size = 10),ncol = 1))

colData <- data.frame(condition = 'A', replicate = 1)

object <- epigraHMMDataSetFromMatrix(countData,colData)

plotCounts(object,ranges = c(500,3500))
```

---

plotPatterns *Create a plot of differerential patterns posterior probabilities from epigraHMM*

Description

'plotPatterns()' plots the posterior probabilities associated with differential patterns from a differential analysis of 'epigraHMM()'

Usage

```
plotPatterns(
  object,
  ranges,
  peaks,
  hdf5 = metadata(object)$output,
  colors = NULL
)
```

Arguments

object	an epigraHMMDataSet
ranges	a GRanges object or a pair of integers with the genomic corrdinates/windows to be plotted
peaks	either a GRanges object or a vector of logicals (with length equal to the number of rows in 'object') specifying the genomic corrdinates/windows with peaks
hdf5	a character string with the hdf5 file path from 'epigraHMM'
colors	an optional argument that specifies the colors for each differential combinatorial pattern

Value

A pheatmat

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
# Creating dummy object
countData <- cbind(rbind(matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1)),
                   rbind(matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1),
                         matrix(rnbinom(1e2, mu = 10, size = 5), ncol = 1),
                         matrix(rnbinom(1e2, mu = 1, size = 10), ncol = 1)))

colData <- data.frame(condition = c('A','B'), replicate = c(1,1))
rowRanges <- GenomicRanges::GRanges('chrA',
                     IRanges::IRanges(start = seq(1,by = 500,
                     length.out = nrow(countData)),width = 500))

object <- epigraHMMDataSetFromMatrix(countData,colData,rowRanges = rowRanges)

# Initializing
object <- initializer(object,controlEM())
```

```
# Running epigraHMM
object <- epigraHMM(object,controlEM(),type = 'differential',dist = 'nb')

# Calling peaks
peaks <- callPeaks(object = object,
                   hdf5 = S4Vectors::metadata(object)$output,
                   method = 'viterbi')

# Plotting patterns
plotPatterns(object,
             ranges = peaks[1],
             peaks = peaks)
```

`segmentGenome` *Segmentation of a genome in non-overlapping windows*

Description

This function segments a genome into non-overlapping windows.

Usage

```
segmentGenome(genome, window, rm.gap = TRUE, rm.blacklist = TRUE)
```

Arguments

`genome`	a string with the name of the genome (e.g. 'hg19')
`window`	an integer with the window size
`rm.gap`	a logical indicating gap regions should be removed
`rm.blacklist`	a logical indicating blacklisted regions should be removed

Value

a GRanges object with the binned genome

Author(s)

Pedro L. Baldoni, <pedrobaldoni@gmail.com>

References

https://github.com/plbaldoni/epigraHMM

Examples

```
gr <- segmentGenome(genome = 'mm10', window = 500)
```

`simulateMarkovChain` *Simulates a Markov Chain of length 'n' given a matrix of transition probabilities P*

Description

Simulates a Markov Chain of length 'n' given a matrix of transition probabilities P

Usage

```
simulateMarkovChain(P, n)
```

Arguments

`P`	a matrix of transition probabilities (row sums should be 1)
`n`	an integer specifying thhe length of the simulated sequence

Examples

```
#Example
simulateMarkovChain(matrix(c(0.99,0.01,0.01,0.99),2,2),100)
```

Index