Package 'segmentSeq'

November 12, 2026

Type Package

Title Methods for identifying small RNA loci from high-throughput sequencing data

Version 2.46.0

Date 2025-07-23

Description
High-throughput sequencing technologies allow the production of large volumes of short sequences, which can be aligned to the genome to create a set of matches to the genome. By looking for regions of the genome which to which there are high densities of matches, we can infer a segmentation of the genome into regions of biological significance. The methods in this package allow the simultaneous segmentation of data from multiple samples, taking into account replicate data, in order to create a consensus segmentation. This has obvious applications in a number of classes of sequencing experiments, particularly in the discovery of small RNA loci and novel mRNA transcriptome discovery.

License GPL-3

LazyLoad yes

Depends R (>= 3.0.0), methods, baySeq (>= 2.9.0), S4Vectors, parallel, GenomicRanges, ShortRead, stats

Suggests BiocStyle, BiocGenerics, knitr, rmarkdown

Imports Rsamtools, IRanges, Seqinfo, graphics, grDevices, utils, abind

biocViews MultipleComparison, Sequencing, Alignment, DifferentialExpression, QualityControl, DataImport

URL https://github.com/samgg/segmentSeq

BugReports https://github.com/samgg/segmentSeq/issues

VignetteBuilder knitr

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

git_branch RELEASE_3_23

git_last_commit b5f4013

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Thomas J. Hardcastle [aut],
Samuel Granjeaud [cre] (ORCID: <https://orcid.org/0000-0001-9245-1535>)

Maintainer Samuel Granjeaud <samuel.granjeaud@inserm.fr>

Contents

segmentSeq-package . . . 3
alignmentClass-class . . . 4
alignmentData-class . . . 5
alignmentMeth-class . . . 7
averageProfiles . . . 8
classifySeg . . . 9
findChunks . . . 11
getCounts . . . 13
getOverlaps . . . 14
givenExpression . . . 16
heuristicSeg . . . 17
hSL . . . 19
lociData-class . . . 20
lociLikelihoods . . . 21
mergeMethSegs . . . 23
methData-class . . . 23
normaliseNC . . . 25
plotGenome . . . 26
plotMeth . . . 27
plotMethDistribution . . . 28
processAD . . . 30
readMethods . . . 31
readMeths . . . 34
segClass-class . . . 35
segData-class . . . 36
segMeth-class . . . 37
selectLoci . . . 38
SL . . . 39
summariseLoci . . . 40
thresholdFinder . . . 40

Index **43**

segmentSeq-package *Segmentation of the genome based on multiple samples of high-throughput sequencing data.*

Description

The segmentSeq package is intended to take multiple samples of high-throughput data (together with replicate information) and identify regions of the genome which have a (reproducibly) high density of tags aligning to them. The package was developed for use in identifying small RNA precursors from small RNA sequencing data, but may also be useful in some mRNA-Seq and chIP-Seq applications.

Details

Package:	segmentSeq
Type:	Package
Version:	0.0.2
Date:	2010-01-20
License:	GPL-3
LazyLoad:	yes
Depends:	baySeq, ShortRead

To use the package, we construct an `alignmentData` object from sets of alignment files using either the `readGeneric` function to read text files or the `readBAM` function to read from BAM format files.

We then use the `processAD` function to identify all potential subsegments of the data and the number of tags that align to these subsegments. We then use either a heuristic or empirical Bayesian approach to segment the genome into 'loci' and 'null' regions. We can then acquire posterior likelihoods for each set of replicates which tell us whether a region is likely to be a locus or a null in that replicate group.

The segmentation is designed to be usable by the `baySeq` package to allow differential expression analyses to be carried out on the discovered loci.

The package (optionally) makes use of the 'snow' package for parallelisation of computationally intensive functions. This is highly recommended for large data sets.

See the vignette for more details.

Author(s)

Thomas J. Hardcastle

Maintainer: Thomas J. Hardcastle <tjh48@cam.ac.uk>

References

Hardcastle T.J., Kelly, K.A. and Balcombe D.C. (2011). Identifying small RNA loci from high-throughput sequencing data. In press.

See Also

baySeq

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an 'alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames)

# Process the alignmentData object to produce a 'segData' object.

sD <- processAD(alignData, gap = 100, cl = NULL)
```

---

alignmentClass-class *Class "alignmentClass"*

Description

The alignmentClass class records information about a set of alignments of high-throughput sequencing data to a genome.

Slots

alignments: Object of class "GRanges". Stores information about the alignments. See Details.

libnames: Object of class "character". The names of the libraries for which alignment data exists.

replicates: Object of class "factor". Replicate information for each of the libraries. See Details.

Details

The alignments slot is a GRanges object defining the location of aligned objects to a reference genome.

The replicates slot is a vector of factors such that the ith sample is a replicate of the jth sample if and only if @replicates[i] == @replicates[j].

The libnames slot is a vector defining the names of the libraries described by the object.

Methods

[`signature(x = "alignmentClass")`: ...

dim `signature(x = "alignmentClass")`: ...

initialize `signature(.Object = "alignmentClass")`: ...

show `signature(object = "alignmentClass")`: ...

Author(s)

Thomas J. Hardcastle

See Also

`alignmentData` `alignmentMeth`

`alignmentData-class` *Class "alignmentData"*

Description

The `alignmentData` class inherits from the `alignmentClass` class and records information about a set of alignments of high-throughput sequencing data to a genome. Details include the alignments themselves, the chromosomes of the genome to which the data are aligned, and counts of the aligned tags from each of the libraries from which the data come.

Objects from the class

Objects can be created by calls of the form `new("alignmentData", ...)`, but more usually by using one of `readBAM` or `readGeneric` functions to generate the object from a set of alignment files.

Slots

`alignments`**:** Object of class `"GRanges"`. Stores information about the alignments. See Details.

`replicates`**:** Object of class `"factor"`. Replicate information for each of the libraries. See Details.

`data`**:** Object of class `"matrix"`. For each alignment described in the `alignments` slot, contains the number of times the alignment is seen in each sample.

`libnames`**:** Object of class `"character"`. The names of the libraries for which alignment data exists.

`libsizes`**:** Object of class `"numeric"`. The library sizes (see Details) for each of the libraries.

Details

The `alignments` slot is the key element of this class. This is a `GRanges` object that, in addition to the usual elements defining the location of aligned objects to a reference genome, also describes the values 'tag', giving the sequence of the tag aligning to the location, 'matches', indicating in how many places that tag matches to the genome, 'chunk', an identifier for the sets of tags that align close enough together to form a potential locus, and 'chunkDup', indicating whether that tag matches to multiple places within the chunk.

The library sizes, defined in the `libsizes` slot, provide some scaling factor for the observed number of counts of a tag in different samples.

The `replicates` slot is a vector of factors such that the ith sample is a replicate of the jth sample if and only if `@replicates[i] == @replicates[j]`.

Methods

[`signature(x = "alignmentData")`: ...

dim `signature(x = "alignmentData")`: ...

initialize `signature(.Object = "alignmentData")`: ...

show `signature(object = "alignmentData")`: ...

Author(s)

Thomas J. Hardcastle

See Also

`alignmentClass`, the class from which `'alignmentData'` inherits. `readGeneric`, which will produce a `'alignmentData'` object from appropriately formatted tab-delimited files. `readBAM`, which will produce a `'alignmentData'` object from BAM files. `processAD`, which will convert an `'alignmentData'` object into a 'segData' object for segmentation.

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an 'alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames)
```

alignmentMeth-class *Class "alignmentMeth"*

Description

The alignmentMeth class inherits from the alignmentClass class and records information about a set of alignments of high-throughput sequencing data to a genome. Details include the alignments themselves, the chromosomes of the genome to which the data are aligned, and counts of the aligned tags from each of the libraries from which the data come.

Objects from the Class

Objects can be created by calls of the form new("alignmentMeth", ...), but more usually by using one of readBAM or readGeneric functions to generate the object from a set of alignment files.

Slots

alignments: Object of class "GRanges". Defines the location of sequenced cytosines, amongst other data. See Details.

libnames: Object of class "character". The names of the libraries for which alignment data exists.

replicates: Object of class "factor". Replicate information for each of the libraries. See Details.

Cs: Object of class "matrix". For each cytosine described in the alignments slot, contains the number of times the cytosine is sequenced as a 'C', and is thus methylated.

Ts: Object of class "matrix". For each cytosine described in the alignments slot, contains the number of times the cytosine is sequenced as a 'T', and is thus unmethylated.

nonconversion: Object of class "numeric". The (estimated) nonconversion rate (see Details) for each of the libraries.

Details

The nonconversion slot is an estimate of the rate (for each library) at which an unmethylated cytosine has failed to be converted by sodium bisulphite treatment into thymine, and is thus recorded (incorrectly) as methylated. In some cases, this can be estimated from considering observed methylation rates on regions known to be unmethylated (e.g., chloroplasts) or by introducing unmethylated control sequences.

The replicates slot is a vector of factors such that the ith sample is a replicate of the jth sample if and only if @replicates[i] == @replicates[j].

Methods

[signature(x = "alignmentMeth"): ...

dim signature(x = "alignmentMeth"): ...

initialize signature(.Object = "alignmentMeth"): ...

show signature(object = "alignmentMeth"): ...

Author(s)

Thomas J. Hardcastle

See Also

alignmentClass, the class from which 'alignmentMeth' inherits. readMeths, which will produce a 'alignmentMeth' object from files generated by the YAMA aligner. processAD, which will convert an 'alignmentMeth' object into a 'segData' object for segmentation.

---

averageProfiles — *Computes and plots the average distribution of aligned reads (taken from an alignmentData object) or methylation (taken from an alignmentMeth object) over a set of coordinates (and optionally the surrounding regions).*

Description

Given an alignmentData or alignmentMeth object and a set of coordinates, plots the average distribution of coverage/methylation over those coordinates. The plotted distributions can be split up into different sample groups by the user.

Usage

```
averageProfiles(mD, samples, coordinates, cuts, maxcuts = 200, bw =
5000, surrounding = 0, add = FALSE, col, ylim, meanOnly = TRUE, plot =
TRUE, ...)
plotAverageProfile(position, profiles, col, surrounding, ylim, add =
FALSE, meanOnly = TRUE, legend = TRUE, titles, ...)
```

Arguments

mD	The alignmentData or alignmentMeth object defining the coverage or methylation on the genome.
samples	A factor or list defining the different groups of samples for which to plot different distributions. If a list, each member should consist of integer values defining the column numbers of the 'mD' object. If missing, will use the mD@replicates value.
coordinates	A GRanges object defining the coordinates of interest (e.g. genic regions).
cuts	Optionally, the number of subdivisions to cut the coordinates in when calculating the average coverage/methylation density.
maxcuts	The maximum number of subdivisions permitted when calculating the average coverage/methylation density.
bw	If 'cuts' is missing, this factor divides the product of the length of the 'coordinates' object and the median width of the coordinates to defines the number of cuts (minimum twenty cuts).

surrounding	If non-zero, the size of the region up- and down-stream of the given coordinates for which average coverage/methylation should also be calculated.
add	If TRUE, the plotted distribution will be added to the current plot.
col	If given, a vector of colours for each of the groups defined in 'samples'. Defaults to 'rainbow(length(samples))'.
ylim	See 'ylim' option for plot. If missing, will be calculated from data.
meanOnly	If TRUE, the mean methylation profile for each member of the 'samples' parameter is plotted on a single graph. If FALSE, every 5th percentile is plotted for each member of the sample parameters, each on a separate graph.
plot	Should the profile be plotted? Defaults to TRUE.
position	A vector describing the position of each point to be plotted. Take from the '$position' element in the list object returned by 'averageProfiles'.
profiles	A matrix describing the profiles to be plotted. Take from the '$profiles element in the list object returned by 'averageProfiles'.
legend	If TRUE, a legend describing the samples is included on the plot.
titles	If given, and 'meanOnly = FALSE', a vector of titles for the quantile plots.
...	Additional arguments to be passed to the 'plotAverageProfile' function, and hence to the 'plot' or 'lines' methods.

Value

Invisibly, a list containing the coordinates of the lines plotted.

Author(s)

Thomas J. Hardcastle

`classifySeg` *A method for defining a genome segment map by an empirical Bayesian classification method*

Description

This function acquires empirical distributions of sequence tag density from an already existing (or heuristically defined) segment map. It uses these to classify potential segments as either segments or nulls in order to define a new (and improved) segment map.

Usage

```
classifySeg(sD, cD, aD, lociCutoff = 0.9, nullCutoff = 0.9, subRegion =
NULL, getLikes = TRUE, lR = FALSE, samplesize = 1e5, largeness = 1e8,
tempDir = NULL, recoverFromTemp = FALSE, cl)
```

Arguments

sD	A segData object derived from the 'aD' object.
cD	A lociData object containing an already existing segmentation map, or NULL.
aD	An alignmentData object.
lociCutoff	The minimum posterior likelihood of being a locus for a region to be treated as a locus.
nullCutoff	The minimum posterior likelihood of being a null for a region to be treated as a null.
subRegion	A data.frame object defining the subregions of the genome to be segmented. If NULL (default), the whole genome is segmented.
getLikes	Should posterior likelihoods for the new segmented genome (loci and nulls) be assessed?
lR	If TRUE, locus and null calls are made on the basis of likelihood ratios rather than posterior likelihoods. Not recommended.
samplesize	The sample size to be used when estimating the prior distribution of the data with the getPriors.NB function.
largeness	The maximum size for a split analysis.
tempDir	A directory for storing temporary files produced during the segmentation.
recoverFromTemp	If TRUE, will attempt to recover the position saved in 'tempDir'. Defaults to FALSE. See Details.
cl	A SNOW cluster object, or NULL. See Details.

Details

This function acquires empirical distributions of sequence tag density from the segmentation map defined by the 'cD' argument (if 'cD' is NULL or missing, then the heuristicSeg function is used to define a segmentation map. It uses these empirical distributions to acquire posterior likelihoods on each potential segment being either a true segment or a null region. These posterior likelihoods are then used to define the segment map.

If recoverFromTemp = TRUE, the function will attempt to recover a crashed position from the temporary files in tempDir. At present, the function assumes you know what you are doing, and will perform no checking that these files are suitable for the specified recovery. Use with caution.

Value

A lociData object, containing the segmentation map discovered.

Author(s)

Thomas J. Hardcastle

References

Hardcastle T.J., Kelly, K.A. and Balcombe D.C. (2011). Identifying small RNA loci from high-throughput sequencing data. In press.

See Also

heuristicSeg a fast heuristic alternative to this function. plotGenome, a function for plotting the alignment of tags to the genome (together with the segments defined by this function). baySeq, a package for discovering differential expression in lociData objects.

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, gap = 100)

# Process the alignmentData object to produce a `segData' object.

sD <- processAD(alignData, gap = 100, cl = NULL)

# Use the classifySeg function on the segData object to produce a lociData object.

pS <- classifySeg(aD = alignData, sD = sD,
subRegion = data.frame(chr = ">Chr1", start = 1, end = 1e5), getLikes = TRUE, cl = NULL)
```

---

findChunks *Identifies 'chunks' of data within a set of aligned reads.*

Description

This function identifies chunks of data within a set of aligned reads by looking for gaps within the alignments; regions where no reads align. If we assume that a locus should not contain a gap of sufficient length, then we can separate the analysis of the data into chunks defined by these gaps, reducing the complexity of the problem of segmentation.

Usage

```
findChunks(alignments, gap, checkDuplication = TRUE, justChunks = FALSE)
```

Arguments

alignments	A GRanges object defining a set of aligned reads.
gap	The minimum length of a gap across which it is assumed that no locus can exist.
checkDuplication	Should we check whether or not reads are duplicated within a chunk? Defaults to TRUE.
justChunks	If TRUE, returns a vector of the chunks rather than the GRanges object with chunks attached. Defaults to FALSE.

Details

This function is called by the readGeneric and readBAM functions but may usefully be called again if filtering of an linkS4class{alignmentData} object has altered the data present, or to increase the computational effort required for subsequent analysis. The lower the 'gap' parameter used to define the chunks, the faster (though potentially less accurate) any subsequent analyses will be.

Value

A modified GRanges object, now containing columns 'chunk' and 'chunkDup' (if 'checkDuplication' is TRUE), identifying the chunk to which the alignment belongs and whether the alignment of the tag is duplicated within the chunk respectively.

Author(s)

Thomas J. Hardcastle

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Read the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, gap = 100)

# Filter the data on number of matches of each tag to the genome

alignData <- alignData[values(alignData@alignments)$matches < 5,]

# Redefine the chunking structure of the data.

alignData <- findChunks(alignData@alignments, gap = 100)
```

getCounts *Gets counts from alignment data from a set of genome segments.*

Description

A function for extracting count data from an alignmentData object given a set of segments defined on the genome.

Usage

```
getCounts(segments, aD, preFiltered = FALSE, adjustMultireads = TRUE,
  useChunk = FALSE, cl)
```

Arguments

segments	A GRanges object which defines a set of segments for which counts are required.
aD	An alignmentData object.
preFiltered	The function internally cleans the data; however, this may not be needed and omitting these steps may save computational time. See Details.
adjustMultireads	If working with methylation data, this option toggles an adjustment for reads that align to multiple locations on the genome. Defaults to TRUE.
useChunk	If all segments are within defined 'chunks' of the alignmentData object, speed increases if this is set to TRUE. Otherwise, counts may be inaccurate. Defaults to FALSE.
cl	A SNOW cluster object, or NULL. See Details.

Details

The function extracts count data from alignmentData object 'aD' given a set of segments. The non-trivial aspect of this function is that at a segment which contains a tag that matches to multiple places in that segment (and thus appears multiple times in the alignmentData object) should count it only once.

If preFiltered = FALSE then the function allows for missing (NA) data in the segments, unordered segments and duplicated segments. If the segment list has no missing data, is already ordered, and contains no duplications, then computational time can be saved by setting preFiltered = TRUE.

A cluster object (package: snow) is recommended for parallelisation of this function when using large data sets. Passing NULL to this variable will cause the function to run in non-parallel mode.

In general, this function will probably not be accessed by the user as the processAD function includes a call to getCounts as part of the standard processing of an alignmentData object into a segData object.

Value

If 'as.matrix', a matrix, each column of which corresponds to a library in the `alignmentData` object 'aD' and each row to the segment defined by the corresponding row in 'segments'. Otherwise an equivalent `DataFrame` object.

Author(s)

Thomas J. Hardcastle

See Also

`processAD`

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an 'alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, gap = 100)

# Get count data for three arbitrarily chosen segments on chromosome 1.

getCounts(segments = GRanges(seqnames = c(">Chr1"),
          IRanges(start = c(1,100,2000), end = c(40,3000,5000))),
          aD = alignData, cl = NULL)
```

---

getOverlaps *Identifies overlaps between two sets of genomic coordinates*

Description

This function identifies which of a set of genomic segments overlaps with another set of coordinates; either with partial overlap or with the segments completely contained within the coordinates. The function is used within the 'segmentSeq' package for various methods of constructing a segmentation map, but may also be useful in downstream analysis (e.g. annotation analyses).

Usage

```
getOverlaps(coordinates, segments, overlapType = "overlapping",
whichOverlaps = TRUE, ignoreStrand = FALSE, cl)
```

Arguments

coordinates	A GRanges object defining the set of coordinates with which the segments may overlap.
segments	A GRanges object defining the set of segments which may overlap within the coordinates.
overlapType	Which kind of overlaps are being sought? Can be one of 'overlapping', 'contains' or 'within'. See Details.
whichOverlaps	If TRUE, returns the 'segments' overlapping with the 'coordinates'. If FALSE, returns a boolean vector specifying which of the 'coordinates' overlap with the 'segments'.
ignoreStrand	If TRUE, a segment may overlap a set of coordinates regardless of the strand the two are on. If FALSE, overlaps will only be identified if both are on the same strand (or if either has no strand specified). Defaults to FALSE.
cl	A SNOW cluster object, or NULL. See Details.

Details

If overlapType = "overlapping" then any overlap between the 'coordinates' and the 'segments' is sufficient. If overlapType = "contains" then a region defined in 'coordinates' must completely contain at least one of the 'segments' to count as an overlap. If overlapType = "within" then a region defined in 'coordinates' must be completely contained by at least one of the 'segments' to count as an overlap.

A 'cluster' object (package: snow) may usefully be used for parallelisation of this function when examining large data sets. Passing NULL to this variable will cause the function to run in non-parallel mode.

Value

If whichOverlaps = TRUE, then the function returns a list object with length equal to the number of rows of the 'coordinates' argument. The 'i'th member of the list will be a numeric vector giving the row numbers of the 'segments' object which overlap with the 'i'th row of the 'coordinates' object, or NA if no segments overlap with this coordinate region.

If whichOverlaps = FALSE, then the function returns a boolean vector with length equal to the number of rows of the 'coordinates' argument, indicating which of the regions defined in coordinates have the correct type of overlap with the 'segments'.

Author(s)

Thomas J. Hardcastle

Examples

```
# Define the chromosome lengths for the genome of interest.

chrlens <- c(2e6, 1e6)

# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 100)

# Find which tags overlap with an arbitrary set of coordinates.

getOverlaps(coordinates = GRanges(seqnames = c(">Chr1"),
          IRanges(start = c(1,100,2000), end = c(40,3000,5000))),
          segments = alignData@alignments, overlapType = "overlapping",
          whichOverlaps = TRUE, cl = NULL)
```

`givenExpression` *Adjusts posterior likelihoods of differential expression by the likelihood that a locus is expressed.*

Description

For certain types of differential behaviour, the likelihood of difference can be high even for regions for which the likelihood that the region represents a true locus is low in all replicate groups. This function corrects the likelihood of differential behaviour by the likelihood that the locus is expressed in at least one replicate group.

Usage

```
givenExpression(cD)
```

Arguments

`cD`	A lociData object with computed posteriors and locLikelihoods.

Value

A lociData object with updated posteriors.

Author(s)

Thomas J. Hardcastle

`heuristicSeg` *A (fast) heuristic method for creation of a genome segment map.*

Description

This method identifies by heuristic methods a set of loci from a `segData` or `segMeth` object. It does this by identifying within replicate groups regions of the genome that satisfy the criteria for being a locus and have no region within them that satisfies the criteria for being a null. These criteria can be defined by the user or inferred from the data.

Usage

```
heuristicSeg(sD, aD, gap = 50, RKPM = 1000, prop, coverage = 1, locCutoff =
0.9, nullCutoff = 0.9, subRegion = NULL, largeness = 1e8, getLikes =
TRUE, verbose = TRUE, tempDir = NULL, cl = NULL, recoverFromTemp =
FALSE, trimMeth = FALSE)
```

Arguments

`aD`	An `alignmentData` or `methData` object.
`sD`	A `segData` or `segMeth` object derived from the 'aD' object.
`gap`	What is the minimum length of a null region?
`RKPM`	For analysis of a `segData` object, what RKPM (reads per kilobase per million reads) distinguishes between a locus and a null region?
`prop`	For analysis of a `segMeth` object, what proportion of methylated cytosines distinguishes between a locus and a null region? (see Details).
`coverage`	For analysis of a `segMeth` object, what is the minimum coverage required to make inferences on the presence/absense of a methylation locus?
`locCutoff`	For analysis of a `segMeth` object, with what likelihood must the proportion of methylated cytosines exceed the 'prop' option to define a locus? Defaults to 0.9.
`nullCutoff`	For analysis of a `segMeth` object, with what likelihood must the proportion of methylated cytosines be less than the 'prop' option to define a null region? Defaults to 0.9.
`subRegion`	A `'data.frame'` object defining the subregions of the genome to be segmented. If NULL (default), the whole genome is segmented.
`largeness`	The maximum size for a split analysis.

getLikes	Should posterior likelihoods for the new segmented genome (loci and nulls) be assessed?
verbose	Should the function be verbose? Defaults to TRUE.
tempDir	A directory for storing temporary files produced during the segmentation.
cl	A SNOW cluster object, or NULL. Defaults to NULL. See Details.
recoverFromTemp	If TRUE, will attempt to recover the position saved in 'tempDir'. Defaults to FALSE. See Details.
trimMeth	Should putative methylation regions be trimmed? Defaults to FALSE; see Details.

Details

A 'cluster' object (package: snow) may be used for parallelisation of parts of this function when examining large data sets. Passing NULL to this variable will cause the function to run in non-parallel mode.

If recoverFromTemp = TRUE, the function will attempt to recover a crashed position from the temporary files in tempDir. At present, the function assumes you know what you are doing, and will perform no checking that these files are suitable for the specified recovery. Use with caution.

The prop variable can be used to set the proportion of methylation required to identify a locus by giving a numerical value between 0-1. It can also be determined automatically (see thresholdFinder).

Due to the way that methylation loci are identified, it is possible that the cytosines at the edges of methylation loci have limited evidence for methylation. The 'trimMeth' option trims cytosines at the edge of predicted methylation loci that have less than 50% likelihood of being above the required threshold.

Value

A lociData object, containing count information on all the segments discovered.

Author(s)

Thomas J. Hardcastle

References

Hardcastle T.J., Kelly, K.A. and Balcombe D.C. (2011). Identifying small RNA loci from high-throughput sequencing data. In press.

See Also

classifySeg, an alternative approach to this problem using an empirical Bayes approach to classify segments. thresholdFinder, a function for determining a suitable threshold on methylation by examining the data. plotGenome, a function for plotting the alignment of tags to the genome (together with the segments defined by this function). baySeq, a package for discovering differential expression in lociData objects.

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, gap = 100)

# Process the alignmentData object to produce a `segData' object.

sD <- processAD(alignData, gap = 100, cl = NULL)

# Use the segData object to produce a segmentation of the genome.

segD <- heuristicSeg(sD = sD, aD = alignData, prop = 0.2,
subRegion = data.frame(chr = ">Chr1", start = 1, end = 1e5),
cl = NULL)
```

---

hSL *Preprocessed 'lociData' object containing likelihoods of methylation at each locus.*

Description

This is the preprocessed (to save computational time) 'lociData' object described in the vignette 'segmentSeq: methods for detecting methylation loci and differential methylation'.

Usage

```
data(hSL)
```

Format

A 'lociData' object.

Source

See vignette.

`lociData-class` *Class "lociData"*

Description

The `lociData` class is based on the `countData` class defined in the 'baySeq' package, but includes a 'coordinates' slot giving the coordinates of genomic loci and a 'locLikelihoods' slot which contains the estimated likelihoods that each annotated region is a locus in each replicate group and a coordinates structure giving the locations of the loci.

Slots

`locLikelihoods`: Object of class `"matrix"` describing estimated likelihoods that each region defined in 'coordinates' is a locus in each replicate group.

`coordinates`: Object of class `"GRanges"` defining the coordinates of the genomic loci.

`data`: Object of class `"matrix"` defining count data for each locus defined in 'coordinates'

`replicates`: Object of class `"factor"` defining the replicate structure of the data.

`groups`: Object of class `"list"` defing the group (model) structure of the data (see the `baySeq` package).

`annotation`: Object of class `"data.frame"` giving any additional annotation information for each locus.

`priorType`: Object of class `"character"` describing the type of prior information available in slot `'priors'`.

`priors`: Object of class `"list"` defing the prior parameter information. Calculated by the baySeq package.

`posteriors`: Object of class `"matrix"` giving the estimated posterior likelihoods for each replicate group. Calculated by the baySeq package.

`nullPosts`: Object of class `"numeric"` which, if calculated, defines the posterior likelihoods for the data having no true expression of any kind. Calculated by the baySeq package.

`estProps`: Object of class `"numeric"` giving the estimated proportion of tags belonnging to each group. Calculated by the baySeq package.

`cellObservables` A list object containing arrays of identical dimension to that in the '@data' slot. These arrays define some observed characteristic of the data (e.g., GC content of sRNAs) which may be used in analysis.

`rowObservables` A list object containing arrays with first dimension identical to the number of rows in the '@data' slot. These arrays define some observed characteristic of the data (e.g., genomic length of the region) which may be used in analysis.

`sampleObservables` A list object containing arrays with first dimension identical to the number of columns of the '@data' slot. These arrays define some observed characteristic of the data (e.g., library scaling factor) which may be used in analysis.

Extends

Class `"countData"`, directly.

Methods

Methods 'new', 'dim', '[', 'c' and 'show' have been defined for this class.

Author(s)

Thomas J. Hardcastle

`lociLikelihoods` *Evaluates the posterior likelihoods of each region defined by a segmentation map as a locus.*

Description

An empirical Bayesian approach that takes a segmentation map and uses this to bootstrap posterior likelihoods on each region being a locus for each replicate group.

Usage

```
lociLikelihoods(cD, aD, newCounts = FALSE, bootStraps = 3,
                inferNulls = TRUE, nasZero = FALSE, usePosteriors =
TRUE, tail = 0.1, subset = NULL, cl)
```

Arguments

`cD`	A `lociData` object that defines a segmentation map.
`aD`	An `alignmentData` object.
`newCounts`	Should new counts be evaluated for the segmentation map in 'cD' before calculating loci likelihoods? Defaults to FALSE
`bootStraps`	What level of bootstrapping should be carried out on the inference of posterior likelihoods? See the baySeq function `getLikelihoods.NB` for a discussion of bootstrapping.
`inferNulls`	Should null regions be inferred from the gaps between segments defined by the 'cD' object?
`nasZero`	If FALSE, any locus with a posterior likelihood 'NA' in the existing segmentation map is treated as a null region for the first bootstrap; If TRUE, it is ignored for the first bootstrap.
`usePosteriors`	If TRUE, the function uses the existing likelihoods to weight the prior estimation of parameters. Defaults to TRUE.
`tail`	The cutoff for the tail of the distribution to be used in pre-calculating data for methylation analysis. See `methObservables`.
`subset`	A subset of the data on which to calculate the likelihoods.
`cl`	A SNOW cluster object, or NULL. See Details.

Details

A 'cluster' object (package: snow) may be used for parallelisation of this function when examining large data sets. Passing NULL to this variable will cause the function to run in non-parallel mode.

Value

A lociData object.

Author(s)

Thomas J. Hardcastle

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, gap = 100)

# Process the alignmentData object to produce a `segData' object.

sD <- processAD(alignData, gap = 100, cl = NULL)

# Use the segData object to produce a segmentation of the genome, but
# without evaluating posterior likelihoods.

segD <- heuristicSeg(sD = sD, aD = alignData,
    subRegion = data.frame(chr= ">Chr1", start = 1, end = 1e5),
    getLikes = FALSE, cl = NULL)

# Use the lociData function to evaluate the posterior likelihoods directly.

lociData <- lociLikelihoods(segD, aD = alignData, bootStraps = 5,
inferNulls = TRUE, cl = NULL)
```

`mergeMethSegs` *Merges neighbouring methylation loci with the same pattern of expression.*

Description

Within a region of cytosine methylation, there may be some cytosines which show no evidence of methylation. The presence of these cytosines may lead to the region being split into multiple methylation loci. This function merges neighbouring loci if the pattern of expression is the same in each locus, and if they are not separated by too great a genomic distance.

Usage

```
mergeMethSegs(segs, aD, gap, cl)
```

Arguments

`segs`	A `methData` object defining the loci to be merged.
`aD`	An `alignmentMeth` object from which the loci have been derived.
`gap`	The maximum gap below which neighbouring loci may be merged.
`cl`	A cluster object, or NULL.

Value

An object of type `methData`.

Author(s)

Thomas J. Hardcastle

See Also

`methData`

`methData-class` *Class "methData"*

Description

The `methData` class is based on the `countData` class defined in the 'baySeq' package, but includes a 'coordinates' slot giving the coordinates of genomic loci and a 'locLikelihoods' slot which contains the estimated likelihoods that each annotated region is a locus in each replicate group and a coordinates structure giving the locations of the loci.

Slots

`locLikelihoods`: Object of class `"matrix"` describing estimated likelihoods that each region defined in 'coordinates' is a locus in each replicate group.

`coordinates`: Object of class `"GRanges"` defining the coordinates of the genomic loci.

`data`: Object of class `"matrix"` defining the number of methylated cytosines observed for each locus defined in 'coordinates'

`data`: Object of class `"matrix"` defining the number of un-methylated cytosines observed for each locus defined in 'coordinates'

`replicates`: Object of class `"factor"` defining the replicate structure of the data.

`groups`: Object of class `"list"` defing the group (model) structure of the data (see the `baySeq` package).

`annotation`: Object of class `"data.frame"` giving any additional annotation information for each locus.

`priorType`: Object of class `"character"` describing the type of prior information available in slot `'priors'`.

`priors`: Object of class `"list"` defing the prior parameter information. Calculated by the baySeq package.

`posteriors`: Object of class `"matrix"` giving the estimated posterior likelihoods for each replicate group. Calculated by the baySeq package.

`nullPosts`: Object of class `"numeric"` which, if calculated, defines the posterior likelihoods for the data having no true expression of any kind. Calculated by the baySeq package.

`estProps`: Object of class `"numeric"` giving the estimated proportion of tags belonnging to each group. Calculated by the baySeq package.

`cellObservables` A list object containing arrays of identical dimension to that in the '@data' slot. These arrays define some observed characteristic of the data (e.g., GC content of sRNAs) which may be used in analysis.

`rowObservables` A list object containing arrays with first dimension identical to the number of rows in the '@data' slot. These arrays define some observed characteristic of the data (e.g., genomic length of the region) which may be used in analysis.

`sampleObservables` A list object containing arrays with first dimension identical to the number of columns of the '@data' slot. These arrays define some observed characteristic of the data (e.g., nonconversion rates) which may be used in analysis.

Extends

Class `"countData"`, directly.

Methods

Methods 'new', 'dim', '[' and 'show' have been defined for this class.

Author(s)

Thomas J. Hardcastle

normaliseNC *A function providing adjustment of cytosine methylated/unmethylated counts based on a nonconversion rate.*

Description

This function adjusts the observed cytosine methylated/unmethylated counts at each cytosine site based on the reported nonconversion rates for each samples.

Usage

```
normaliseNC(mD, nonconversion)
```

Arguments

mD	Either an alignmentMeth or segMeth object, or a lociData object (for which nonconversion must be explicitly supplied).
nonconversion	A vector defining nonconversion rates for each sample, required if a lociData object is supplied in 'mD' and ignored otherwise.

Details

This function operates by estimating the expected number of unconverted cytosines at each site and subtracting this from the reported methylated cytosines and adding to the reported unmethylated cytosines. It should not be used on data that will be analysed in a way that accounts for nonconversion; e.g., using the 'bbNCDist' densityFunction object.

Value

A modified version of the object supplied as 'mD'.

Author(s)

Thomas J. Hardacastle

References

Hardcastle T.J. Discovery of methylation loci and analyses of differential methylation from replicated high-throughput sequencing data. bioRxiv (http://dx.doi.org/10.1101/021436)

Examples

```
datadir <- system.file("extdata", package = "segmentSeq")
files <- c("short_18B_C24_C24_trim.fastq_CG_methCalls.gz",
"short_Sample_17A_trimmed.fastq_CG_methCalls.gz",
"short_13_C24_col_trim.fastq_CG_methCalls.gz",
"short_Sample_28_trimmed.fastq_CG_methCalls.gz")
```

```
mD <- readMeths(files = files, dir = datadir,
libnames = c("A1", "A2", "B1", "B2"), replicates = c("A","A","B","B"),
nonconversion = c(0.004777, 0.005903, 0.016514, 0.006134))

mD <- normaliseNC(mD)
```

`plotGenome` *Plots the alignment of sequence tags on the genome given an 'alignmentData' object and (optionally) a set of segments found.*

Description

Plots the data from an `alignmentData` object for a given set of samples. Can optionally include in the plot the annotation data from a `lociData` object containing segment information.

Usage

```
plotGenome(aD, loci, chr = 1, limits = c(0, 1e4), samples = NULL,
plotType = "pileup", plotDuplicated = FALSE, density = 0, showNumber =
TRUE, logScale = FALSE, cap = Inf, ...)
```

Arguments

`aD`	An `alignmentData` object.
`loci`	A `lociData` object (produced by the `heuristicSeg` or `classifySeg` function and therefore) containing appropriate annotation information. Can be omitted if this annotation is not known/required.
`chr`	The name of the chromosome to be plotted. Should correspond to a chromosome name in the `alignmentData` object.
`limits`	The start and end point of the region to be plotted.
`samples`	The sample numbers of the samples to be plotted. If NULL, plots all samples.
`plotType`	The manner in which the plot is created. Currently only `plotType = pileup` is recommended.
`plotDuplicated`	If TRUE, then any duplicated sequence tags (i.e., sequence tags that match to multiple places in the genome) in the 'aD' object will be plotted on a negative scale for each sample. Defaults to FALSE (recommended).
`density`	The density of the shading lines to be used in plotting each segment.
`showNumber`	Should the row number of each segment be shown?
`logScale`	Should a log scale be used for the number of sequence tags found at each base?
`cap`	A numeric value defining a cap on the maximum number of reads to be plotted at any one point. Useful if a large number of reads at one location prevent a clear signal being seen elsewhere.
`...`	Any additional graphical parameters for passing to `plot`.

Value

Plotting function.

Author(s)

Thomas J. Hardcastle

See Also

alignmentData, heuristicSeg, classifySeg

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, gap = 100)

# Plot the alignments to the genome on chromosome 1 between bases 1 and 10000

plotGenome(alignData, chr = ">Chr1", limits = c(1, 1e5))
```

---

plotMeth *Plots a map of cytosine methylation (and optionally, methylation loci).*

Description

This function takes an alignmentMeth object and plots the observed levels of methylation within some defined region on the genome. If a methData object is also supplied, loci of methylation will also be shown.

Usage

```
plotMeth(aM, loci, chr, limits, samples, showNumber = TRUE, rgb = c(1, 0, 0),
angle = 45, cap, add = FALSE)
```

Arguments

aM	An alignmentMeth.
loci	A lociData object (produced by the heuristicSeg or classifySeg function and therefore) containing appropriate annotation information. Can be omitted if this annotation is not known/required.
chr	The name of the chromosome to be plotted. Should correspond to a chromosome name in the alignmentMeth object.
limits	The start and end point of the region to be plotted.
samples	The sample numbers of the samples to be plotted. If NULL, plots all samples.
showNumber	Should the row number of each segment be shown?
rgb	The rgb code (rgb) with which to colour the loci.
angle	The angle at which loci are shaded (see rect).
cap	Caps the maximum level of coverage shown on the plot; thus, if a base has been sequenced at a level greater than the cap, the data for that base will be shown as if it has a coverage of cap.
add	If TRUE, adds the plot of methylation level to the current plot. Defaults to FALSE.

Value

Plotting function.

Author(s)

Thomas J. Hardcastle

See Also

alignmentMeth

---

plotMethDistribution *Plots the distribution of methylation on the genome.*

Description

Plots the distribution of methylation (as defined in an alignmentMeth object upon the genome.

Usage

```
plotMethDistribution(meth, samples, bw = 1e-3, subtract, chrs, centromeres,
add = FALSE, col, ylim = NULL, legend = TRUE, ...)
```

Arguments

meth	An object of class alignmentMeth containing the methylation data.
samples	A numeric vector defining the columns of data in the 'meth' object from which to estimate proportions of methylation, or a list object containing numeric vectors if multiple distributions are to be derived from the 'meth' object, or a factor in which each level of the factor defines a set of columns for the 'meth' object. If missing, derived from the '@replicates' slot of the 'meth' object.
bw	Gives the bandwidth of the density plots; analagous to the 'bw' parameter in density.
subtract	A numeric vector giving values to be subtracted from the density of methylation. See Details.
chrs	The names of the chromosomes for which a distribution should be plotted. If mising, derived from the '@alignments' slot of the 'meth' object.
centromeres	If given, a numeric vector defining the position of the centromeres on the chromosomes. These will be then be plotted.
add	Should the distribution curve be added to an existing plot? Defaults to FALSE.
col	A vector of colours to be used to plot the distributions. If missing, generated from rainbow.
ylim	Limits on the y-axis. Defaults to NULL, in which case limits are automatically set.
legend	Should a legend be added to the plot? Defaults to TRUE.
...	Any additional parameters to be passed to plot.

Details

The function returns the density of methylation calculated. This can be used in further plots as the 'subtract' parameter, which allows one methylation profile to be subtracted from another.

Value

An object of class density describing the plotted distribution.

Author(s)

Thomas J. Hardcastle

See Also

alignmentMeth

processAD *Processes an 'alignmentData' or 'alignmentMeth' object into a 'segData' or 'segMeth' object for segmentation.*

Description

In order to discover segments of the genome with a high density of sequenced data, a 'segData' object must be produced. This is an object containing a set of potential segments, together with the counts for each sample in each potential segment.

Usage

```
processAD(aD, gap = 300, squeeze = 2, filterProp = 0.05, strandSplit = FALSE,
verbose = TRUE, getCounts = FALSE, cl)
```

Arguments

aD	An alignmentData or alignmentMeth object.
gap	The maximum gap between aligned tags that should be allowed in constructing potential segments. Defaults to 300. See Details.
squeeze	If greater than zero, the minimum gap between aligned tags that should be allowed in constructing potential segments. See Details.
filterProp	If 'aD' is a alignmentMeth object and this is given, the minimum proportion of methylation at a base below which the base will be filtered out before constructing potential segments (but not during counting).
strandSplit	If TRUE, the data will be split by strand and segments will be constructed separately for each strand. Defaults to FALSE.
verbose	Should processing information be displayed? Defaults to TRUE.
getCounts	If TRUE, counts will be estimated for the constructed 'segData' object. If FALSE, they will not, and must be estimated on the fly for further operations on the 'segData' object, which is computationally wasteful but will substantially reduce the memory requirements.
cl	A SNOW cluster object, or NULL. See Details.

Details

This function takes an alignmentData or alignmentMeth object and constructs a segData or segMeth object from it. The function creates a set of potential segments by looking for all locations on the genome where the start of a region of overlapping alignments (or, if 'squeeze' is non-zero, those alignments separated by no more than 'squeeze'.) exists in the alignmentData object. A potential segment then exists from this start point to the end of all regions of overlapping alignments such that there is no region in the segment of at least length 'gap' where no tag aligns. The number of potential segments can therefore be increased by increasing this limit, or (usually more usefully) decreased by decreasing this limit in order to save computational effort.

A 'cluster' object (package: snow) is recommended for parallelisation of this function when using large data sets. Passing NULL to this variable will cause the function to run in non-parallel mode.

Value

A segData object.

Author(s)

Thomas J. Hardcastle

See Also

getCounts, which produces the count data for each potential segment. heuristicSeg and classifySeg, which segment the genome based on the segData object produced by this function segData alignmentData

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, gap = 100)

# Process the alignmentData object to produce a `segData' object.

sD <- processAD(alignData, gap = 100, cl = NULL)
```

---

readMethods *Functions for processing files of various formats into an 'alignmentData' object.*

Description

These functions take alignment files of various formats to produce an object (see Details) describing the alignment of sequencing tags from different libraries. At present, BAM and text files are supported.

Usage

```
readGeneric(files, dir = ".", replicates, libnames, chrs, chrlens, cols,
            header = TRUE, minlen = 15, maxlen = 1000,
            multireads = 1000, polyLength, estimationType = "quantile",
            discardTags = FALSE, verbose = TRUE, filterReport = NULL, ...)

readBAM(files, dir = ".", replicates, libnames, chrs, chrlens, countID = NULL,
        minlen = 15, maxlen = 1000, multireads = 1000,
     polyLength, estimationType = "quantile", discardTags = FALSE, verbose = TRUE,
        filterReport = NULL)
```

Arguments

files	Filenames of the files to be read in.
dir	Directory (or directories) in which the files can be found.
replicates	A vector defining the replicate structure if the group. If and only if the ith library is a replicate of the jth library then `@replicates[i] == @replicates[j]`. This argument may be given in any form but will be stored as a factor.
libnames	Names of the libraries defined by the file names.
chrs	A chracter vector defining (a selection of) the chromosome names used in the alignment files; optional, will be inferred from data if not given.
chrlens	Lengths of the chromosomes to which the alignments were made; optional, will be inferred from data if not given.
cols	A named character vector which describes which column of the input files contains which data. See Details.
countID	A (two-character) string used by the BAM file to identify the 'counts' of individual sequenced reads; that is, how many times a given read appears in the sequenced library. If NULL, it is assumed that the data are redundant (see Details).
header	Do the input files have a header line? Defaults to TRUE. See Details.
minlen	Minimum length for reads.
maxlen	Maximum length for reads.
multireads	The functions will discard any read that aligns to the genome in more locations than given by this value. Set to Inf to keep everything. Defaults to 1000.
polyLength	If given, an integer value N defining the length of (approximate) homopolymers which will be removed from the data. If a tag contains a sequence of N+1 reads consisting of at least N identical bases, it will be removed. If not given, all data is used.
estimationType	The estimationType that will be used by the 'baySeq' function `getLibsizes` to infer the library sizes of the samples.
discardTags	If TRUE, information about the sequence of the aligned reads will be discarded. Useful for very large data sets. Defaults to FALSE.
verbose	Should processing information be displayed? Defaults to TRUE.

filterReport	If not NULL, this should be a string defining a file to which will be written those data filtered on the basis of chromsome choices, widths of sequences, multireads or polyBase.
...	Additional parameters to be passed to read.table. In particular, the 'sep' and 'skip' arguments may be useful.

Details

readBAM: This function takes a set of BAM files and generates the 'alignmentData' object from these. If a character string for 'countID' is given, the function assumes the data are non-redundant and that 'countID' identifies the count data (i.e., how many times each read appears in the sequenced library) in each BAM file. If 'countID' is NULL, then it is assumed that the data are redundant, and the count data are inferred from the file.

readGeneric: The purpose of this function is to take a set of plain text files and produce an 'alignmentData' object. The function uses read.table to read in the columns of data in the files and so by default columns are separated by any white space. Alternative separators can be used by passing the appropriate value for 'sep' to read.table.

The files may contain columns with column names 'chr', 'tag', 'count', 'start', 'end', 'strand' in which case the 'cols' argument can be ommitted and 'header' set to TRUE. If this is the case, there is no requirement for all the files to have the same ordering of columns (although all must have these column names).

Alternatively, the columns of data in the input files can be specified by the 'cols' argument in the form of a named character vector (e.g; 'cols = c(chr = 1, tag = 2, count = 3, start = 4, end = 5, strand = 6)' would cause the function to assume that the first column contains the chromosome information, the second column contained the tag information, etc. If 'cols' is specified then information in the header is ignored. If 'cols' is missing and 'header' is FALSE, then it is assumed that the data takes the form described in the example above.

The 'tag', 'count' and 'strand' columns may optionally be omitted from either the file column headers or the 'cols' argument. If the 'tag' column is omitted, then the data will not account for duplicated sequences when estimating the number of counts in loci. If the 'count' column is omitted, the 'readGeneric' function will assume that the file contains the alignments of each copy of each sequence tag, rather than an aggregated alignment of each unique sequence. The unique alignments will be identified and the number of sequence tags aligning to each position will be calculated. If 'strand' is omitted, the strand will simply be ignored.

Value

An alignmentData object.

Author(s)

Thomas J. Hardcastle

See Also

alignmentData

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an `alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames)
```

`readMeths` *A function for reading data from the YAMA methylation aligner (or similarly parsed data) from which to identify methylation loci and/or differentially methylated regions.*

Description

This function takes as input a set of files that describe the number of times a set of cytosines are observed to be methylated or unmethylated in some high-throughput sequencing data. It merges the data from these files into an object of '`alignmentMeth`' class which can then be further processed to identify methylation loci.

Usage

```
readMeths(files, dir = ".", libnames, replicates, nonconversion, chrs)
```

Arguments

`files`	A character vector defining the file names of the alignment files to be read in.
`dir`	The directory in which the files are located.
`libnames`	A character vector giving the names of the samples to be read in.
`replicates`	A vector defining the replicate structure of the data. The 'i'th and 'j'th libraries are treated as replicates if and only if replicates[i] == replicates[j].
`nonconversion`	A numeric vector (all members should lie between 0 and 1) defining the non-conversion rate of each library. See `alignmentMeth-class` for details.
`chrs`	An (optional) character vector giving the names of the chromosomes to be read from the files. If ommitted, all chromosomes will be read in.

Value

An object of class alignmentMeth.

Author(s)

Thomas J. Hardcastle

See Also

alignmentMeth-class.

Examples

```
datadir <- system.file("extdata", package = "segmentSeq")
files <- c("short_18B_C24_C24_trim.fastq_CG_methCalls.gz",
"short_Sample_17A_trimmed.fastq_CG_methCalls.gz",
"short_13_C24_col_trim.fastq_CG_methCalls.gz",
"short_Sample_28_trimmed.fastq_CG_methCalls.gz")

mD <- readMeths(files = files, dir = datadir,
libnames = c("A1", "A2", "B1", "B2"), replicates = c("A","A","B","B"),
nonconversion = c(0.004777, 0.005903, 0.016514, 0.006134))
```

---

segClass-class *Class "segClass"*

Description

The segClass class contains data about potential segments on the genome.

Objects from the class

Objects can be created by calls of the form new("segClass", ..., seglens). However, more usually they will be created by calling the processAD function.

Slots

coordinates: A GRanges object defining the coordinates of the segments.

replicates: Object of class "factor". The replicate structure for the samples.

locLikelihoods: Object of class "DataFrame" describing estimated likelihoods that each region defined in 'coordinates' is a locus in each replicate group.

Details

The @coordinates slot contains information on each of the potential segments; specifically, chromosome, start and end of the segment, together. Each row of the @coordinates slot should correspond to the same row of the @data slot.

In almost all cases objects of this class should be produced by the processAD function.

Methods

Methods 'new', 'dim', '[' and 'show' have been defined for this class.

Author(s)

Thomas J. Hardcastle

See Also

processAD, the function that will most often be used to create objects of this class. segData, which inherits from this class. segMeth, which inherits from this class.

---

segData-class *Class "segData"*

Description

The segData class inherits from the segClass class and contains data about potential segments on the genome, together with counts for each of those segments.

Objects from the class

Objects can be created by calls of the form new("segData", ..., seglens). However, more usually they will be created by calling the processAD function.

Slots

coordinates: A GRanges object defining the coordinates of the segments.

replicates: Object of class "factor". The replicate structure for the samples.

locLikelihoods: Object of class "DataFrame" describing estimated likelihoods that each region defined in 'coordinates' is a locus in each replicate group.

data: Object of class matrix. Contains the number of counts observed for each sample in each potential segment.

libsizes: Object of class "numeric". The library sizes for each sample.

Details

The @coordinates slot contains information on each of the potential segments; specifically, chromosome, start and end of the segment, together. Each row of the @coordinates slot should correspond to the same row of the @data slot.

In almost all cases objects of this class should be produced by the processAD function.

Methods

Methods 'new', 'dim', '[' and 'show' have been defined for this class.

Author(s)

Thomas J. Hardcastle

See Also

processAD, the function that will most often be used to create objects of this class. classifySeg, an empirical Bayesian method for defining a segmentation based on a segData object.

Examples

```
# Define the files containing sample information.

datadir <- system.file("extdata", package = "segmentSeq")
libfiles <- c("SL9.txt", "SL10.txt", "SL26.txt", "SL32.txt")

# Establish the library names and replicate structure.

libnames <- c("SL9", "SL10", "SL26", "SL32")
replicates <- c(1,1,2,2)

# Process the files to produce an 'alignmentData' object.

alignData <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames)

# Process the alignmentData object to produce a 'segData' object.

sD <- processAD(alignData, gap = 100, cl = NULL)
```

---

segMeth-class *Class "segMeth"*

Description

The segMeth class inherits from the segClass class and contains data about potential segments on the genome, together with counts for each of those segments.

Objects from the class

Objects can be created by calls of the form new("segMeth", ..., seglens). However, more usually they will be created by calling the processAD function.

Slots

coordinates: A GRanges object defining the coordinates of the segments.

replicates: Object of class "factor". The replicate structure for the samples.

`locLikelihoods`: Object of class `"DataFrame"` describing estimated likelihoods that each region defined in 'coordinates' is a locus in each replicate group.

`Cs`: Object of class `matrix`. Contains the number of methylated cytosines (which are sequenced as a 'C') observed for each sample in each potential segment.

`Ts`: Object of class `matrix`. Contains the number of unmethylated cytosines (which are sequenced as a 'T') observed for each sample in each potential segment.

`nonconversion`: Object of class `"numeric"`. The (estimated) nonconversion rate (see Details) for each of the libraries.

Details

The `@coordinates` slot contains information on each of the potential segments; specifically, chromosome, start and end of the segment, together. Each row of the `@coordinates` slot should correspond to the same row of the `@C` and `@T` slots.

The nonconversion slot is an estimate of the rate (for each library) at which an unmethylated cytosine has failed to be converted by sodium bisulphite treatment into thymine, and is thus recorded (incorrectly) as methylated. In some cases, this can be estimated from considering observed methylation rates on regions known to be unmethylated (e.g., chloroplasts) or by introducing unmethylated control sequences.

In almost all cases objects of this class should be produced by the `processAD` function.

Methods

Methods 'new', 'dim', '[' and 'show' have been defined for this class.

Author(s)

Thomas J. Hardcastle

See Also

`processAD`, the function that will most often be used to create objects of this class. `segClass`, from which this class inherits.

---

selectLoci *Filters a 'lociData' object based on given selection criteria.*

Description

Selects loci from a 'lociData' object based on likelihood, false discovery rate or familywise error rate for downstream processing.

Usage

```
selectLoci(cD, likelihood, FDR, FWER, perReplicate = TRUE)
```

Arguments

cD	The lociData object to be filtered.
likelihood	If provided, all loci with a likelihood greater than this criterion will be selected.
FDR	If provided (and likelihood is not provided), the maximal set of loci which controls the FDR at this level is selected.
FWER	If provided (and likelihood and FDR are not provided), the maximal set of loci which controls the FWER at this level is selected.
perReplicate	If TRUE, selection of loci is done on a replicate by replicate basis. If FALSE, selection will be done on the likelihood that the locus represents a true locus in at least one replicate group.

Value

A lociData object.

Author(s)

Thomas J. Hardcastle

See Also

lociLikelihoods

SL *Example data selected from a set of Illumina sequencing experiments.*

Description

Each of the files 'SL9', 'SL10', 'SL26' and 'SL32' represents a subset of the data from an Illumina sequencing experiment. These data consist of alignment information; the tag sequence, and the number of times that each sequence is observed.

Format

A set of tab-delimited files containing data from four sequencing experiments.

Source

In-house Illumina sequencing experiments

summariseLoci *Summarise the expected number of loci in a 'lociData' object.*

Description

Summarises the expected number of loci, either in toto or on a per replicate group basis.

Usage

```
summariseLoci(cD, perReplicate = TRUE)
```

Arguments

cD	A 'lociData' object with calculated values in the '@lociLikelihoods' slot.
perReplicate	Should the expectation be calculated on a per replicate group basis, or the total number of loci identified in the dataset?

Value

A numeric vector summarising the expectated number of loci in the cD object.

Author(s)

Thomas J. Hardcastle

thresholdFinder *Determines threshold for the proportion of methylation at which a methylation locus may be identified.*

Description

This function offers a variety of methods for the analysis of methylation data to determine a suitable threshold for the proportion of methylation at which to distinguish a methylation locus from a non-methylated locus.

Usage

```
thresholdFinder(method, aM, subset, minprop = 0.05, bootstrap = 100,
abstol = 1e-4, verbose = FALSE, cl = NULL, processAD.args = list(),
heuristicSeg.args = list())
```

Arguments

method	Character string defining method to use for threshold estimation. Available options are 'varsum', 'minden', 'beta' and 'abc'. See Details.
aM	An alignmentMeth object containing observed methylation counts.
subset	Numeric vector defining a subset on aM object for use in threshold estimation.
minprop	For 'minden' method, a minimum proportion permitted for choice of threshold.
bootstrap	The maximum number of bootstraps to be permitted in estimating a threshold. Defaults to 100. See Details.
abstol	Minimum tolerance fro threshold estimation.
verbose	Verbose reporting. Defaults to FALSE.
cl	A cluster object, or NULL. Defaults to NULL.
processAD.args	Arguments to be passed to processAD function if bootstrapping.
heuristicSeg.args	Arguments to be passed to heuristicSeg function if bootstrapping.

Details

This function operates on the data observed within each replicate group, and then takes the mean of the thresholds calculated for each group.

Methods currently available for threshold estimation are 'varsum', 'minden', 'beta' and 'abc'. The 'varsum' method attempts to split the vector of proportions of methylation observed at each cytosine into two sets of minimal total variance. The 'minden' method finds the minimum point on a smoothed kernel density of the proportions of methylation. The 'beta' method estimates for each cytosine a posterior distribution on proportions of methylation based on the beta-binomial conjugacy, takes the average of these distributions and finds the minumum. The 'abc' method performs like the beta method, but estimates the posterior distribution through approximate Bayesian computation.

Bootstrapping uses the estimated threshold to define loci. Based on the defined loci, cytosines are then only included in a re-estimation of the thresholds if they are identified as belonging to an expressed locus within the current replicate group, or if they are not expressed in any replicate group. Thresholds are re-esimated until the maximum number of bootstraps is reached or the difference between estimated thresholds drops below 'abstol', whichever is the sooner.

Value

A numeric value defining a threshold on methylation.

Author(s)

Thomas J. Hardcastle

See Also

heuristicSeg

Examples

```
datadir <- system.file("extdata", package = "segmentSeq")
files <- c("short_18B_C24_C24_trim.fastq_CG_methCalls.gz",
"short_Sample_17A_trimmed.fastq_CG_methCalls.gz",
"short_13_C24_col_trim.fastq_CG_methCalls.gz",
"short_Sample_28_trimmed.fastq_CG_methCalls.gz")

mD <- readMeths(files = files, dir = datadir,
libnames = c("A1", "A2", "B1", "B2"), replicates = c("A","A","B","B"),
nonconversion = c(0.004777, 0.005903, 0.016514, 0.006134))

## Not run: thresholdFinder("beta", mD, cl = NULL)
```

Index

∗ **classes**
alignmentClass-class, 4
alignmentData-class, 5
alignmentMeth-class, 7
lociData-class, 20
methData-class, 23
segClass-class, 35
segData-class, 36
segMeth-class, 37
∗ **classif**
classifySeg, 9
heuristicSeg, 17
∗ **datasets**
hSL, 19
SL, 39
∗ **files**
readMethods, 31
readMeths, 34
∗ **hplot**
averageProfiles, 8
plotGenome, 26
plotMeth, 27
plotMethDistribution, 28
∗ **mainip**
givenExpression, 16
summariseLoci, 40
∗ **manip**
classifySeg, 9
findChunks, 11
getCounts, 13
getOverlaps, 14
heuristicSeg, 17
lociLikelihoods, 21
mergeMethSegs, 23
normaliseNC, 25
processAD, 30
selectLoci, 38
thresholdFinder, 40
∗ **package**
segmentSeq-package, 3
[,alignmentClass,ANY,ANY,ANY-method *(alignmentClass-class)*, 4
[,alignmentClass,ANY,ANY-method *(alignmentClass-class)*, 4
[,alignmentClass,ANY-method *(alignmentClass-class)*, 4
[,alignmentClass-method *(alignmentClass-class)*, 4
[,alignmentData,ANY,ANY,ANY-method *(alignmentData-class)*, 5
[,alignmentData,ANY,ANY-method *(alignmentData-class)*, 5
[,alignmentData,ANY-method *(alignmentData-class)*, 5
[,alignmentData-method *(alignmentData-class)*, 5
[,alignmentMeth,ANY,ANY,ANY-method *(alignmentMeth-class)*, 7
[,alignmentMeth,ANY,ANY-method *(alignmentMeth-class)*, 7
[,alignmentMeth,ANY-method *(alignmentMeth-class)*, 7
[,alignmentMeth-method *(alignmentMeth-class)*, 7
[,lociData,ANY,ANY,ANY-method *(lociData-class)*, 20
[,lociData,ANY,ANY-method *(lociData-class)*, 20
[,lociData,ANY-method *(lociData-class)*, 20
[,lociData-method *(lociData-class)*, 20
[,methData,ANY,ANY,ANY-method *(methData-class)*, 23
[,methData,ANY,ANY-method *(methData-class)*, 23
[,methData,ANY-method *(methData-class)*, 23
[,methData-method *(methData-class)*, 23

[,segClass,ANY,ANY-method
(segClass-class), 35
[,segClass-method (segClass-class), 35
[,segData,ANY,ANY-method
(segData-class), 36
[,segData,ANY-method (segData-class), 36
[,segData-method (segData-class), 36
[,segMeth,ANY,ANY-method
(segMeth-class), 37
[,segMeth,ANY-method (segMeth-class), 37
[,segMeth-method (segMeth-class), 37

alignmentClass, *6*, *8*
alignmentClass (alignmentClass-class), 4
alignmentClass-class, 4
alignmentData, *3*, *5*, *8*, *10*, *13*, *17*, *21*, *26, 27*, *30, 31*, *33*
alignmentData (alignmentData-class), 5
alignmentData-class, 5
alignmentMeth, *5*, *8*, *23*, *25*, *27–30*, *34, 35*, *41*
alignmentMeth (alignmentMeth-class), 7
alignmentMeth-class, 7
averageProfiles, 8

baySeq, *3, 4*, *11*, *18*, *20*, *24*

c,lociData-method (lociData-class), 20
cbind,alignmentClass-method
(alignmentClass-class), 4
cbind,alignmentData-method
(alignmentData-class), 5
cbind,alignmentMeth-method
(alignmentMeth-class), 7
classifySeg, 9, *18*, *26–28*, *31*, *37*
countData, *20*, *23, 24*

density, *29*
dim,alignmentClass-method
(alignmentClass-class), 4
dim,alignmentData-method
(alignmentData-class), 5
dim,alignmentMeth-method
(alignmentMeth-class), 7
dim,lociData-method (lociData-class), 20
dim,methData-method (methData-class), 23
dim,segClass-method (segClass-class), 35
dim,segData-method (segData-class), 36
dim,segMeth-method (segMeth-class), 37

findChunks, 11

getCounts, 13, *31*
getLibsizes, *32*
getLikelihoods.NB, *21*
getOverlaps, 14
getPriors.NB, *10*
givenExpression, 16
GRanges, *8*, *12*, *35–37*

heuristicSeg, *10, 11*, 17, *26–28*, *31*, *41*
hSL, 19

initialize,alignmentClass-method
(alignmentClass-class), 4
initialize,alignmentData-method
(alignmentData-class), 5
initialize,alignmentMeth-method
(alignmentMeth-class), 7
initialize,segClass-method
(segClass-class), 35
initialize,segData-method
(segData-class), 36
initialize,segMeth-method
(segMeth-class), 37

lociData, *10, 11*, *18*, *21, 22*, *25, 26*, *28*, *39*
lociData (lociData-class), 20
lociData-class, 20
lociLikelihoods, 21, *39*

mergeMethSegs, 23
methData, *17*, *23*, *27*
methData (methData-class), 23
methData-class, 23
methObservables, *21*

normaliseNC, 25

plot, *29*
plotAverageProfile (averageProfiles), 8
plotGenome, *11*, *18*, 26
plotMeth, 27
plotMethDistribution, 28
processAD, *3*, *6*, *8*, *13, 14*, 30, *35–38*

rainbow, *29*
read.table, *33*
readBAM, *3*, *5–7*, *12*
readBAM (readMethods), 31
readGeneric, *3*, *5–7*, *12*
readGeneric (readMethods), 31

readMethods, 31
readMeths, *8*, 34
rect, *28*
rgb, *28*

segClass, *38*
segClass *(*segClass-class*)*, 35
segClass-class, 35
segData, *10*, *17*, *30*, *31*, *36*
segData *(*segData-class*)*, 36
segData-class, 36
segmentSeq *(*segmentSeq-package*)*, 3
segmentSeq-package, 3
segMeth, *17*, *25*, *30*, *36*
segMeth *(*segMeth-class*)*, 37
segMeth-class, 37
selectLoci, 38
show,alignmentClass-method
*(*alignmentClass-class*)*, 4
show,alignmentData-method
*(*alignmentData-class*)*, 5
show,alignmentMeth-method
*(*alignmentMeth-class*)*, 7
show,lociData-method *(*lociData-class*)*,
20
show,methData-method *(*methData-class*)*,
23
show,segClass-method *(*segClass-class*)*,
35
show,segData-method *(*segData-class*)*, 36
show,segMeth-method *(*segMeth-class*)*, 37
SL, 39
SL10 *(*SL*)*, 39
SL26 *(*SL*)*, 39
SL32 *(*SL*)*, 39
SL9 *(*SL*)*, 39
summariseLoci, 40

thresholdFinder, *18*, 40