segmentSeq

October 25, 2011

`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.

Usage

```
SL
```

Format

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

Source

In-house Illumina sequencing experiments

`alignmentData-class` *Class "alignmentData"*

Description

The `alignmentData` class records information about a set of alignments of high-throughput sequencing data to a genome. Details include the alignments themselves, details on the chromosomes of the genome to which the data are aligned, and information on 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 the `processTags` function.

Slots

`alignments`: Object of class `"data.frame"`. Stores information about the alignments. 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.

`chrs`: Object of class `"data.frame"`. Should contain two columns 'chr' and 'len' giving the chromosome names and lengths of each chromosome respectively.

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

Details

The `alignments` slot is the key element of this class. This is a `"data.frame"` object that contains the columns 'chr', 'start', 'end', 'duplicated', 'tag', 'count', 'sampleNumber' and 'replicate'. Columns 'chr', 'start' and 'end' define the chromosome, start and end point of the tag. 'duplicated' indicates whether or not the tag uniquely matches this location (`FALSE`) or whether the tag matches some other location on the genome (`TRUE`). The 'tag' column gives the sequence of the tag as a factor. The 'count' column gives the number of times the tag appears in the library. Which library is involved is specified by the 'sampleNumber' column, and the 'replicate' column gives the replicate group that this library is associated with.

The library sizes, defined in the `libsizes` slot, provide some scaling factor for the observed number of counts of a tag in different samples. One method of calculating this, for example, would be to take the number of sequences read from the high-throughput sequencing machine that align to the reference genome.

The `replicates` slot should take the form of a vector of integers such that if and only if the ith sample is a replicate of the jth sample then `@replicates[i] == @replicates[j]`. In addition, values in the replicates slot should take values from `1:n` where `n` is the number of replicate groups.

Methods

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

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

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

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

Note

Methods 'new', 'dim', '[', 'cbind' and 'show' have been defined for these classes.

Author(s)

Thomas J. Hardcastle

See Also

processTags, which will produce a 'alignmentData' object from appropriately formatted tab-delimited files. processAD, which will convert an 'alignmentData' object into a 'segData' object for segmentation.

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens)
```

---

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

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, cl, ...)
```

Arguments

sD	A segData object derived from the 'aD' object.
cD	A countData 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.
cl	A SNOW cluster object, or NULL. See Details.
...	Any additional parameters to be passed to heuristicSeg.

Details

This function acquires empirical distributions of sequence tag density from the segmentation map defined by the 'cD' argument (if 'cD = NULL' 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.

Value

A postSeg object, containing the segmentation map discovered.

Author(s)

Thomas J. Hardcastle

References

Hardcastle T.J., and Kelly, K.A. (2010). Genome Segmentation From High-Throughput Sequencing Data. In preparation.

See Also

heuristicSeg 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 countData objects.

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

sD <- processAD(alignData, cl = NULL)

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

pS <- classifySeg(aD = alignData, sD = sD, subRegion = data.frame(chr = ">Chr1", start =
```

`filterSegments` *Filters an set of segments (given an ordering on the segments) such*

Description

This function takes a set of segments, plus an ordering on that set, and filters the set such that no segments overlap, preferentially keeping the segments first in ordering.

Usage

```
filterSegments(segs, orderOn, ...)
```

Arguments

`segs`	A `'data.frame'` containing columns 'chr', 'start' and 'end'.
`orderOn`	An vector of some statistic that can be used to create an ordering on the 'segs' data.frame.
`...`	Additional parameters that can be passed to `order` when ordering the segments using the 'orderOn' parameter.

Details

This function takes the set of segments defined by the `data.frame` 'segs', together with some statistic (e.g., likelihood of similarity with background) defined by the 'orderOn' vector. Additional options can be passed to the 'order' function (for example, relating to the direction of the ordering) through the '...' parameter.

The function takes the segment first in the ordering and discards any segments that overlap with it. It then proceeds to the next remaining segment in the ordering and discards any segments that overlap with this. This process continues until we have a set of non-overlapping segments.

This function can be used to create a random sample of non-overlapping segments by providing a randomly chosen set of values for the 'orderOn' vector.

Value

A vector giving the rows of the `data.frame` object 'segs' which form a non-overlapping set.

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 <- dir(datadir, pattern = ".txt", full.names = TRUE)

# Establish the library names and replicate structure.

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

sD <- processAD(alignData, cl = NULL)

# Create random sampling of non-overlapping segments for chromosome 1 of sD object.

filterSegments(subset(sD@segInfo, select = c(chr, start, end)), runif(nrow(sD)))
```

`findChunks` *Identifies 'chunks' of data within an 'alignmentData' object.*

Description

This function identifies chunks of data within an 'alignmentData' object by looking for gaps within the alignments; regions where no tags 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(aD, gap)
```

Arguments

aD	An alignmentData object.
gap	The minimum length of a gap across which it is assumed that no locus can exist.

Details

This function is called by the processTags function 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 any subsequent analyses will be.

Value

A modified alignmentData object, in which the '@alignments' slot contains columns 'chunk' and 'chunkDup', 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 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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

alignData <- alignData[alignData@alignments$matches < 5,]

# Redefine the chunking structure of the data.

alignData <- findChunks(alignData, gap = 200)
```

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, cl)
```

Arguments

segments	A 'data.frame' 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.
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

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 the data.frame 'segments'.

Author(s)

Thomas J. Hardcastle

See Also

processAD

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

getCounts(segments = data.frame(chr = ">Chr1", 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 =
```

Arguments

`coordinates`	A `data.frame` object, with columns 'chr', 'start' and 'end', defining the set of coordinates with which the segments may overlap.
`segments`	A `data.frame` object, with columns 'chr', 'start' and 'end', 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'.

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 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' data.frame which overlap with the 'i'th row of the 'coordinates' data.frame, 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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

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

`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' 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, bimodality = TRUE, RKPM = 30, gap = 100, subRegion
= NULL, getLikes = TRUE, verbose = TRUE, cl)
```

Arguments

`aD`	An `alignmentData` object.
`sD`	A `segData` object derived from the 'aD' object.
`bimodality`	Should the criteria for loci be inferred from the (likely) bimodaly structure of the data?
`RKPM`	What RKPM (reads per kilobase per million reads) distinguishes between a locus and a null region? Ignored if bimodality = TRUE.
`gap`	What is the minimum length of a null region? Ignored if bimodality = TRUE.
`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?
`verbose`	Should the function be verbose? Defaults to TRUE.
`cl`	A SNOW cluster object, or NULL. 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.

Value

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

Author(s)

Thomas J. Hardcastle

References

Hardcastle T.J., and Kelly, K.A. (2010). Genome Segmentation From High-Throughput Sequencing Data. In preparation.

See Also

classifySeg, an alternative approach to this problem using an empirical Bayes approach to classify segments. 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 countData objects.

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

sD <- processAD(alignData, cl = NULL)

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

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

---

lociLikelihoods *Evaluates the posterior likelihoods of each region defined by a*

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 = 1,
                inferNulls = TRUE, nasZero = FALSE, cl)
```

Arguments

cD	A countData or postSeg 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 getLikelihoods.NB.
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.
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 postSeg object.

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
```

```
chrlens, gap = 200)

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

sD <- processAD(alignData, 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 postSeg function to evaluate the posterior likelihoods directly.

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

`mergeSD` *Merges multiple 'segData' objects.*

Description

This file takes two or more segData objects, which may have different co-ordinates defined in the '@segInfo' slot and merges them into one for subsequent analysis. This is not quite equivalent to 'cbind' as it may change the co-ordinate structure of the data.

Usage

```
mergeSD(..., aD, replicates, gap = 200, cl = NULL)
```

Arguments

`...`	Two or more 'segData' objects.
`aD`	An 'alignmentData' object that describes (in the same order) all samples that appear in the 'segData' objects described in '...'
`replicates`	The replicate structure of the new 'segData' object'.
`gap`	The maximum gap between aligned tags that should be allowed in constructing potential segments. See Details.
`cl`	A SNOW cluster object, or NULL.

Value

A 'segData' object.

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 first two files to produce an 'alignmentData' object.

alignData12 <- processTags(file = libfiles[1:2], dir = datadir, replicates =
replicates[1:2], libnames = libnames[1:2], chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

# Process the alignmentData object to produce a 'segData' object.
sD12 <- processAD(alignData12, cl = NULL)

# Repeat with the third and fourth files

alignData34 <- processTags(file = libfiles[3:4], dir = datadir, replicates =
replicates[3:4], libnames = libnames[3:4], chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)
sD34 <- processAD(alignData34, cl = NULL)

# Bind the two alignData objects together
alignData <- cbind(alignData12, alignData34)

# Merge the two segData objects
sD <- mergeSD(sD12, sD34, aD = alignData, replicates = c(1,1,2,2), gap =200, cl = NULL)

# Note missing data in new 'segData' object.
sD
```

`plotGenome` *Plots the alignment of sequence tags on the genome given an*

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 `countData` object containing segment information.

Usage

```
plotGenome(aD, sD, chr = 1, limits = c(0, 1e4), samples = NULL,
plotType = "chunk", plotDuplicated = FALSE, ...)
```

Arguments

aD	An alignmentData object.
sD	A countData 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 (translated into 'character' type if given in any other form) 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 supported.
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.
...	Any additional graphical parameters for passing to plot.

Value

Plotting function.

Author(s)

Thomas J. Hardcastle

See Also

alignmentData, heuristicSeg, classifySeg

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
```

```
chrlens, gap = 200)

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

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

---

postSeg-class *baySeq - classes*

Description

The postSeg class is identical to the countData class defined in the 'baySeq' package, but has a different structure in the '@posteriors' slot and so is given a different class to avoid confusion.

Slots

Objects of the 'postSeg' class should contain the following components:

data:	Count data (matrix).
libsizes:	Vector of library size for each sample.
groups:	Group (model) structure to test on the data (list).
annotation:	Annotation data for each count (data.frame).
priorType:	Character string describing the type of prior information available in slot 'priors'.
priors:	Prior parameter information.
posteriors:	Estimated posterior likelihoods for each replicate group of a potential segment being a null (matrix).
estProps:	Estimated proportion of tags belonging to each group (numeric). Calculated by the functions described
nullPosts:	If calculated, the posterior likelihoods for the data having no true expression of any kind.
seglens:	Lengths of segments containing the counts described in data. A matrix, but may be initialised with a v

Details

The seglens slot describes, for each row of the data object, the length of the 'segment' that contains the number of counts described by that row. For example, if we are looking at the number of hits matching genes, the seglens object would consist of transcript lengths. Exceptionally, we may want to use different segment lengths for different samples and so the slot takes the form of a matrix. If the matrix has only one column, it is duplicated for all samples. Otherwise, it should have the same number of columns as the '@data' slot. If the slot is the empty matrix, then it is assumed that all segments have the same length.

Methods

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

Author(s)

Thomas J. Hardcastle

processAD *Processes an 'alignmentData' object into a 'segData' object for*

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 = NULL, verbose = TRUE, cl)
```

Arguments

aD	An alignmentData object.
gap	The maximum gap between aligned tags that should be allowed in constructing potential segments. See Details.
verbose	Should processing information be displayed? Defaults to TRUE.
cl	A SNOW cluster object, or NULL. See Details.

Details

This function takes an alignmentData object and constructs a segData 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 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 'gap' argument thus defines the maximum gap that can exist between tags in a segment of high density of alignments. 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.

The 'gap' argument is now by default specified in the processTags function used to create the 'aD' object, and so 'gap' can be left as NULL providing this has been done.

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 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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

sD <- processAD(alignData, gap = 200, cl = NULL)
```

`processTags` *Convenience function for processing tab-delimited files in a certain*

Description

This function takes files in a text format with defined columns (see Details) that describe the alignment of sequencing tags from different libraries.

Usage

```
processTags(files, dir = ".", replicates, libnames, chrs, chrlens, cols, header
TRUE, gap = 200, verbose = TRUE, ...)
```

Arguments

`files`	Filenames of the files to be read in.
`dir`	Directory (or directories) in which the files can be found.
`replicates`	Replicate information on the libraries. See Details.
`libnames`	Names of the libraries defined by the file names.
`chrs`	Chromosome names (as `'character'`) used in the alignment files.
`chrlens`	Lengths of the chromosomes to which the alignments were made.
`cols`	A named character vector which describes which column of the input files contains which data. See Details.
`header`	Do the input files have a header line? Defaults to TRUE. See Details.

gap	The maximum gap between aligned tags that should be allowed in constructing potential segments. See Details.
verbose	Should processing information be displayed? Defaults to TRUE.
...	Additional parameters to be passed to read.table.

Details

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', 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)' would cause the function to assume that the first column contains the chromosome information, the second column contained the tag information, &c. If 'cols' is specified then information in the header is ignored. If 'cols' is missing and 'header = FALSE' then it is assumed that the data takes the form described in the example above.

The 'tag' and 'count' 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 'processTags' 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.

The replicates argument should take the form of a vector of integers such that if and only if the ith library is a replicate of the jth library then @replicates[i] == @replicates[j]. In addition, values in the replicates slot should take values from 1:n where n is the number of replicate groups.

Value

An alignmentData object.

Author(s)

Thomas J. Hardcastle

See Also

alignmentData

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)
```

`segData-class` *Class "segData"*

Description

The `segData` class contains data about potential segments on the genome containing data about each potential subsegment.

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

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

`leftData`: Object of class `"matrix"`. Contains the number of counts observed for the region to the left of the potential segment.

`rightData`: Object of class `"matrix"`. Contains the number of counts observed for the region to the right of the potential segment.

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

`chrs`: Object of class `"data.frame"`. Should contain two columns 'chr' and 'len' giving the chromosome names and lengths of each chromosome respectively.

`replicates`: Object of class `"numeric"`. The replicate structure for the samples. This should be a vector of consecutive integers starting with 1.

`priorType`: Character string describing the type of prior information available in slot `'priors'`.

`priors`: Prior parameter information, estimated from the data (or otherwise acquired). See Details.

`segInfo`: Object of class `"data.frame"`. A data.frame containing the following columns; 'chr', 'start', 'end', 'leftSpace', 'rightSpace'. See Details.

Details

The `@segInfo` slot contains information on each of the potential segments; specifically, chromosome, start and end of the segment, together with the distance from each segment to the next segment on the left and right hand sides. These data are contained in the columns 'chr', 'start', 'end', 'leftSpace', 'rightSpace' respectively. Each row of the `@segInfo` 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 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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

sD <- processAD(alignData, cl = NULL)

# Estimate prior parameters for the segData object.
```

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

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.

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 (either explicitly or using the `processTags` function). containing the alignment information for each sample. 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 empirically determine the prior parameters of the data using the `getPriors` function, and finally identify all segments to which a high density of tags align in at least one replicate group using the `segmentSeq` function. The output from this segmentation is designed to be usable by the `baySeq` package.

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., and Kelly, K.A. (2010). Genome Segmentation from High-Throughput Sequencing Data. In submission.

See Also

`baySeq`

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 <- processTags(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, gap = 200)

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

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

Index