# Package ‘segmentSeq’

September 24, 2012

**Type** Package

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

**Version** 1.8.0

**Date** 2010-01-20

**Author** Thomas J. Hardcastle

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

**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 (>= 2.3.0), methods, baySeq (>= 1.8.1), ShortRead,GenomicRanges, IRanges

**Suggests** snow

**Imports** baySeq, graphics, grDevices, IRanges, methods, utils,GenomicRanges

**Collate** AllClasses.R segData-accessors.R alignmentData-accessors.R
postSeq-accessors.R getCounts.R plotGenome.R processAD.R
processTags.R getOverlaps.R heuristicSeg.R classifySeg.R
lociLikelihoods.R processPosteriors.R findChunks.R utilityFunctions.R

**biocViews** Bioinformatics, HighThroughputSequencing,MultipleComparisons

# R topics documented:

---

`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 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)

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

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

## 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 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.

**`data`:** Object of class `"DataFrame"`. 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.

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

## 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

`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 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)
```

## `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, 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.

`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.

## 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 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)

# 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 lociData object.

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

---

`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)
```

## 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.

## 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 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)

# Read 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)

# 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, as.matrix = 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. |
| `as.matrix` | If TRUE, returns the counts as a matrix. Otherwise, returns the counts as a `DataFrame`. |
| `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 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)

# 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, 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'. |
| 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)
```

---

## `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, RKPM = 1000, gap = 100, subRegion
= NULL, largeness = 1e8, getLikes = TRUE, verbose = TRUE, cl)
```

### Arguments

| | |
|---|---|
| `aD` | An `alignmentData` object. |
| `sD` | A `segData` object derived from the ‘aD’ object. |

`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.

`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.

`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 `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. `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 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)

# 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)
```

## `lociData-class` *Class "lociData"*

### Description

The `lociData` class is based on the `lociData` 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.

`libsizes`: Object of class `"numeric"` describing the library size (scaling factor) for each sample.

`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.

`seglens`: Object of class `"matrix"` defining the lengths of each segment containing the counts described in the `'data'` slot. May be initialised with a vector, or ignored altogether.

### Extends

Class "lociData", directly.

### 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

---

## 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 = 1,
                inferNulls = TRUE, nasZero = FALSE, usePosteriors =
TRUE, 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. |
| `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 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)

# 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 lociData function to evaluate the posterior likelihoods directly.

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

## `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, locData, 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.

`locData` 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 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)

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

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

# processAD *Processes an ‘alignmentData’ object into a ‘segData’ 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 = 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 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 readGeneric and readBAM functions 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 <- readGeneric(file = libfiles, dir = datadir, replicates =
replicates, libnames = libnames, chrs = c(">Chr1", ">Chr2"), chrlens =
chrlens, 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, gap = 200, polyLength, estimationType = "quantile",
            verbose = TRUE, ...)

readBAM(files, dir = ".", replicates, libnames, chrs, chrlens, countID = NULL,
        gap = 200, polyLength, estimationType = "quantile", verbose = TRUE)
```

### 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.

`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.

`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.

`gap` The maximum gap between aligned tags that should be allowed in constructing potential segments. See `findChunks`.

`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.

`verbose` Should processing information be displayed? Defaults to TRUE.

`...` 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,  stran
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 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)
```

## 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 DataFrame. Contains the number of counts observed for each sample in each potential segment.

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

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

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

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

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

sD <- processAD(alignData, cl = NULL)

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

---

## 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

# Index