Package ‘girafe’

January 20, 2025

Type Package

Title Genome Intervals and Read Alignments for Functional Exploration

Version 1.58.0

Date 2024-04-23

Depends R (>= 2.10.0), methods, BiocGenerics (>= 0.13.8), S4Vectors (>= 0.17.25), Rsamtools (>= 1.31.2), intervals (>= 0.13.1), ShortRead (>= 1.37.1), genomeIntervals (>= 1.25.1), grid

Imports methods, Biobase, Biostrings (>= 2.47.6), pwalign, graphics, grDevices, stats, utils, IRanges (>= 2.13.12)

Suggests MASS, org.Mm.eg.db, RColorBrewer

Enhances genomeIntervals

Author Joern Toedling, with contributions from Constance Ciaudo, Olivier Voinnet, Edith Heard, Emmanuel Barillot, and Wolfgang Huber

Maintainer J. Toedling `<jtoedling@yahoo.de>`

Description The package 'girafe' deals with the genome-level representation of aligned reads from next-generation sequencing data. It contains an object class for enabling a detailed description of genome intervals with aligned reads and functions for comparing, visualising, exporting and working with such intervals and the aligned reads. As such, the package interacts with and provides a link between the packages ShortRead, IRanges and genomeIntervals.

License Artistic-2.0

LazyLoad yes

biocViews Sequencing

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

git_branch RELEASE_3_20

git_last_commit ff75045

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-01-20

Contents

`Additional reduce-methods`

Auxiliary methods for Function reduce in Package 'girafe'

Description

This methods were written to resurrect the functionality of the 'reduce' method of package 'IRanges' for objects belonging to classes defined in 'IRanges'. This functions had been overwritten by the later import of package 'intervals'. See the corresponding help pages of package IRanges for more details on these methods.

Methods

`signature(x = "CompressedIRangesList")` see help page in package 'IRanges'

`signature(x = "IRanges")` see help page in package 'IRanges'

`signature(x = "IntegerRanges")` see help page in package 'IRanges'

`signature(x = "IntegerRangesList")` see help page in package 'IRanges'

See Also

`IntegerRangesList-class`, `IntegerRanges-class`, `IRanges-class`

`addNBSignificance` *assess significance of sliding-window read counts*

Description

This function can be used to assess the significance of sliding-window read counts. The background distribution of read counts in windows is assumed to be a Negative-Binomial (NB) one. The two parameters of the NB distribution, mean 'mu' and dispersion 'size', are estimated using any of the methods described below (see details). The estimated NB distribution is used to assign a p-value to each window based on the number of aligned reads in the window. The p-values can be corrected for multiple testing using any of the correction methods implemented for `p.adjust`.

Usage

```
addNBSignificance(x, estimate="NB.012", correct = "none",  max.n=10L)
```

Arguments

`x`	A `data.frame` of class `slidingWindowSummary`, as returned by the function `perWindow`.
`estimate`	string; which method to use to estimate the parameters of the NB background distribution; see below for details
`correct`	string; which method to use for p-value adjustment; can be any method that is implemented for `p.adjust` including "none" if no correction is desired.
`max.n`	integer; only relevant if `estimate=="NB.ML"`; in that case specifies that windows with up to this number of aligned reads should be considered for estimating the background distribution.

Details

The two parameters of the Negative-Binomial (NB) distribution are: mean 'λ' (or 'mu') and size 'r' (or 'size').

The function knows a number of methods to estimate the parameters of the NB distribution.

"NB.012" Solely the windows with only 0, 1, or 2 aligned reads are used for estimating λ and 'r'. From the probability mass function $g(k) = P(X = k)$ of the NB distribution, it follows that the ratios

$$q_1 = \frac{g(1)}{g(0)} = \frac{\lambda \cdot r}{\lambda + r}$$

and

$$q_2 = \frac{g(2)}{g(1)} = \frac{\lambda \cdot (r+1)}{2 \cdot (\lambda + r)} .$$

The observed numbers of windows with 0-2 aligned reads are used to estimate

$$\widehat{q_1} = \frac{n_1}{n_0}$$

and

$$\widehat{q_2} = \frac{n_2}{n_1}$$

and from these estimates, one can obtain estimates for $\widehat{\lambda}$ and $\widehat{r}$.

“NB.ML” This estimation method uses the function `fitdistr` from package ‘MASS’. Windows with up to `n.max` aligned reads are considered for this estimate.

“Poisson” This estimate also uses the windows the 0-2 aligned reads, but uses these numbers to estimates the parameter λ of a Poisson distribution. The parameter ‘r’ is set to a very large number, such that the estimated NB distribution actually is a Poisson distribution with mean and variance equal to λ.

Value

A `data.frame` of class `slidingWindowSummary`, which is the the supplied argument `x` extended by an additional column `p.value` which holds the p-value for each window. The attribute `NBparams` of the result contains the list of the estimated parameters of the Negative-Binomial background distribution.

Author(s)

Joern Toedling

References

Such an estimation of the Negative-Binomial parameters has also been described in the paper:
Ji et al.(2008) An integrated system CisGenome for analyzing ChIP-chip and ChIP-seq data. Nat Biotechnol. 26(11):1293-1300.

See Also

`perWindow`, `p.adjust`

Examples

```
exDir <- system.file("extdata", package="girafe")
exA   <- readAligned(dirPath=exDir, type="Bowtie",
  pattern="aravinSRNA_23_no_adapter_excerpt_mm9_unmasked.bwtmap")
exAI  <- as(exA, "AlignedGenomeIntervals")
exPX  <- perWindow(exAI, chr="chrX", winsize=1e5, step=0.5e5)
exPX  <- addNBSignificance(exPX, correct="bonferroni")
str(exPX)
exPX[exPX$p.value <= 0.05,]
```

`agiFromBam` *Create AlignedGenomeIntervals objects from BAM files.*

Description

Function to create `AlignedGenomeIntervals` objects from BAM (binary alignment map format) files. Uses functions from package `Rsamtools` to parse BAM files.

Usage

```
agiFromBam(bamfile, ...)
```

Arguments

`bamfile`	File path of BAM file. BAM file should be sorted and have an index in the same directory (see Details below).
`...`	further arguments passed on to function `scanBam`

Details

Note: the BAM files must be sorted and must also have an index file (`*.bai`) in the same directory. These should be done when creating the BAM. However, the functions `sortBam` and `indexBam` can be used for the same purpose, as can the respective modules of the “samtools” library (‘samtools sort’ and ‘samtools index’).

The BAM files are parsed chromosome by chromosome to limit the memory footprint of the function. Thus, this function aims to be a less-memory-consuming alternative to first reading in the BAM file using the `readAligned` function and then converting the `AlignedRead` object into an `AlignedGenomeIntervals` object.

Value

An object of class `AlignedGenomeIntervals`.

Author(s)

J Toedling

References

`http://samtools.sourceforge.net`

See Also

`scanBam`, `AlignedGenomeIntervals-class`

Examples

```
fl <- system.file("extdata", "ex1.bam", package="Rsamtools")
ExGi <- agiFromBam(fl)
head(detail(ExGi))
```

`AlignedGenomeIntervals-class`
Class 'AlignedGenomeIntervals'

Description

A class for representing reads from next-generation sequencing experiments that have been aligned to genomic intervals.

Objects from the Class

Objects can be created either by:

1. calls of the form `new("AlignedGenomeIntervals", .Data, closed, ...)`.
2. using the auxiliary function `AlignedGenomeIntervals` and supplying separate vectors of same length which hold the required information:
`AlignedGenomeIntervals(start, end, chromosome, strand, reads, matches, sequence)`
If arguments `reads` or `matches` are not specified, they are assumed to be '1' for all intervals.
3. or, probably the most common way, by coercing from objects of class `AlignedRead`.

Slots

`.Data`: two-column integer matrix, holding the start and end coordinates of the intervals on the chromosomes

`sequence`: character; sequence of the read aligned to the interval

`reads`: integer; total number of reads that were aligned to this interval

`matches`: integer; the total number of genomic intervals that reads which were aligned to this interval were aligned to. A value of '1' thus means that this read sequence matches uniquely to this one genome interval only

`organism`: string; an identifier for the genome of which organism the intervals are related to. Functions making use of this slot require a specific annotation package `org.<organism>.eg.db`. For example if `organism` is 'Hs', the annotation package 'org.Hs.eg.db' is utilised by these functions. The annotation packages can be obtained from the Bioconductor repositories.

`annotation`: data.frame; see class `genome_intervals` for details

`closed`: matrix; see class `genome_intervals` for details

`type`: character; see class `genome_intervals` for details

`score`: numeric; optional score for each aligned genome interval

`id`: character; optional identifier for each aligned genome interval

`chrlengths`: integer; optional named integer vector of chromosome lengths for the respective genome; if present it is used in place of the chromosome lengths retrieved from the annotation package (see slot `organism`)

Extends

Class `Genome_intervals-class`, directly. Class `Intervals_full`, by class "Genome_intervals", distance 2.

Methods

coerce Coercion method from objects of class `AlignedRead`, which is defined in package `ShortRead`, to objects of class `AlignedGenomeIntervals`

coverage `signature("AlignedGenomeIntervals")`: computes the read coverage over all chromosomes. If the `organism` of the object is set correctly, the chromosome lengths are retrieved from the appropriate annotation package, otherwise the maximum interval end is taken to be the absolute length of that chromosome (strand).
The result of this method is a list and the individual list elements are of class `Rle`, a class for encoding long repetitive vectors that is defined in package `IRanges`.
The additional argument `byStrand` governs whether the coverage is computed separately for each strand. If `byStrand=FALSE` (default) only one result is returned per chromosome. If

`byStrand=TRUE`, there result is two separate `Rle` objects per chromosome with the strand appended to the chromosome name.

detail `signature("AlignedGenomeIntervals")`: a more detailed output of all the intervals than provided by `show`; only advisable for objects containing few intervals

extend `signature("AlignedGenomeIntervals")` with additional arguments `fiveprime=0L` and `threeprime=0L`. These must be integer numbers and greater than or equal to 0. They specify how much is subtracted from the left border of the interval and added to the right side. Which end is 5' and which one is 3' are determined from the strand information of the object. Lastly, if the object has an `organism` annotation, it is checked that the right ends of the intervals do not exceed the respective chromosome lengths.

export export the aligned intervals as tab-delimited text files which can be uploaded to the UCSC genome browser as 'custom tracks'. Currently, there are methods for exporting the data into 'bed' format and 'bedGraph' format, either writing the intervals from both strands into one file or into two separate files (formats 'bedStrand' and 'bedGraphStrand', respectively). Details about these track formats can be found at the UCSC genome browser web pages.

The additional argument `writeHeader` can be set to `FALSE` to suppress writing of the track definition header line to the file.

For `Genome_intervals` objects, only 'bed' format is supported at the moment and does not need to be specified.

hist `signature("AlignedGenomeIntervals")`: creates a histogram of the lengths of the reads aligned to the intervals

organism Get or set the organism that the genome intervals in the object correspond to. Should be a predefined code, such as 'Mm' for mouse and 'Hs' for human. The reason for this code, that, if the organism is set, a corresponding annotation package that is called `org.<organism>.eg.db` is used, for example for obtaining the chromosome lengths to be used in methods such as `coverage`. These annotation packages can be obtained from the Bioconductor repository.

plot visualisation method; a second argument of class `Genome_intervals_stranded` can be provided for additional annotation to the plot. Please see below and in the vignette for examples. Refer to the documentation of `plotAligned` for more details on the plotting function.

reduce collapse/reduce aligned genome intervals by combining intervals which are completely included in each other, combining overlapping intervals AND combining immediately adjacent intervals (if `method="standard"`). Intervals are only combined if they are on the same chromosome, the same strand AND have the same match specificity of the aligned reads.

If you only want to combine intervals that have exactly the same start and stop position (but may have reads of slightly different sequence aligned to them), then use the argument `method="exact"`.

If you only want to combine intervals that have exactly the same 5' or 3' end (but may differ in the other end and in the aligned sequence), then use the argument `method="same5"` (same 5' end) or `method="same3"` (same 3' end).

Finally, it's possible to only collapse/reduce aligned genome intervals that overlap each other by at least a certain fraction using the argument `min.frac`. `min.frac` is a number between 0.0 and 1.0. For example, if you call `reduce` with argument `min.frac=0.4`, only intervals that overlap each other by at least 40 percent are collapsed/merged.

sample draw a random sample of `n` (Argument `size`) of the aligned reads (without or with replacement) and returns the `AlignedGenomeIntervals` object defined by these aligned reads.

score access or set a custom score for the object

sort sorts the intervals by chromosome name, start and end coordinate in increasing order (unless `decreasing=TRUE` is specified) and returns the sorted object

subset take a subset of reads, matrix-like subsetting via '\[' can also be used

Author(s)

Joern Toedling

See Also

`Genome_intervals-class`, `AlignedRead-class`, `plotAligned`

Examples

```
############# toy example:
A <- new("AlignedGenomeIntervals",
       .Data=cbind(c(1,3,4,5,8,10), c(5,5,6,8,9,11)),
       annotation=data.frame(
         seq_name=factor(rep(c("chr1","chr2","chr3"), each=2)),
         strand=factor(c("-","-","+","+","+","+") ,levels=c("-","+")),
         inter_base=rep(FALSE, 6)),
       reads=rep(3L, 6), matches=rep(1L,6),
       sequence=c("ACATT","ACA","CGT","GTAA","AG","CT"))

show(A)
detail(A)

## alternative initiation of this object:
A <- AlignedGenomeIntervals(
   start=c(1,3,4,5,8,10), end=c(5,5,6,8,9,11),
   chromosome=rep(c("chr2","chrX","chr1"), each=2),
   strand=c("-","-","+","+","+","+"),
   sequence=c("ACATT","ACA","CGT","GGAA","AG","CT"),
   reads=c(1L, 5L, 2L, 7L, 3L, 3L))
detail(A)

## custom identifiers can be assigned to the intervals
id(A) <- paste("gi", 1:6, sep="")

## subsetting and combining
detail(A[c(1:4)])
detail(c(A[1], A[4]))

## sorting: always useful
A <- sort(A)
detail(A)

## the 'reduce' method provides a cleaned-up, compact set
detail(reduce(A))
##  with arguments specifying additional conditions for merging
detail(reduce(A, min.frac=0.8))

## 'sample' to draw a sample subset of reads and their intervals
detail(sample(A, 10))

## biological example
exDir <- system.file("extdata", package="girafe")
exA   <- readAligned(dirPath=exDir, type="Bowtie",
  pattern="aravinSRNA_23_no_adapter_excerpt_mm9_unmasked.bwtmap")
exAI <- as(exA, "AlignedGenomeIntervals")
organism(exAI) <- "Mm"
```

```
show(exAI)
## which chromosomes are the intervals on?
table(chromosome(exAI))

## subset
exAI[is.element(chromosome(exAI),  c("chr1","chr2"))]

## compute coverage per chromosome:
coverage(exAI[is.element(chromosome(exAI),  c("chr1","chr2"))])

### plotting:
load(file.path(exDir, "mgi_gi.RData"))
if (interactive())
   plot(exAI, mgi.gi, chr="chrX", start=50400000, end=50410000)

### overlap with annotated genome elements:
exOv <- interval_overlap(exAI, mgi.gi)
## how many elements do read match positions generally overlap:
table(listLen(exOv))
## what are the 13 elements overlapped by a single match position:
mgi.gi[exOv[[which.max(listLen(exOv))]]]
## what kinds of elements are overlapped
(tabOv <- table(as.character(mgi.gi$type)[unlist(exOv)]))
### display those classes:
my.cols <- rainbow(length(tabOv))
if (interactive())
   pie(tabOv, col=my.cols, radius=0.85)
```

`countReadsAnnotated` *Sum up aligned reads per category of genome feature*

Description

A function to sum up aligned reads per category of genome feature (i.e. gene, ncRNA, etc.).

Usage

```
countReadsAnnotated(GI, M, typeColumn="type", fractionGI=0.7,
                      mem.friendly=FALSE, showAllTypes=FALSE)
```

Arguments

`GI`	object of class `AlignedGenomeIntervals`
`M`	Annotation object of class `Genome_intervals_stranded` or `Genome_intervals`; describes the genomic coordinates of annotated genome features, such as genes, miRNAs, etc.
`typeColumn`	string; which column of the annotation object `M` describes the type of the genome feature
`fractionGI`	which fraction of the intervals in object `GI` are required to ovelap with a feature in `M` in order to be considered to correspond to that feature.
`mem.friendly`	logical; should a version which requires less memory but takes a bit longer be used
`showAllTypes`	logical; should a table of genome feature types in `M` be displayed?

Details

The read counts are summed up over each type of genome feature, and the read counts are normalised by their number of genomic matches. For example if a read has two matches in the genome, but only one inside a miRNA, it would count 0.5 for miRNAs.

Value

A named numeric vector which gives the summed read counts for each supplied type of genome feature.

Author(s)

J Toedling

Examples

```
A <- AlignedGenomeIntervals(
         start=c(1,8,14,20), end=c(5,15,19,25),
         chromosome=rep("chr1", each=4),
         strand=c("+","+","+","+"),
         sequence=c("ACATT","TATCGGAC","TCGGACT","GTAACG"),
         reads=c(7L, 2L, 4L, 5L) )
M2 <- new("Genome_intervals_stranded",
          rbind(c(2,6), c(1,15), c(20,30)),
          closed = matrix(TRUE, ncol=2, nrow=3),
          annotation = data.frame(
            seq_name= factor(rep("chr1", 3)),
            inter_base= logical(3),
            strand=factor(rep("+", 3), levels=c("+","-")),
            alias=c("miRNA1","gene1","tRNA1"),
            type=c("miRNA","gene","tRNA")) )
if (interactive()){
   grid.newpage()
   plot(A, M2, chr="chr1", start=0, end=35,
        nameColum="alias", show="plus")
}
countReadsAnnotated(A, M2, typeColumn="type")
```

`fracOverlap` *Retrieve intervals overlapping by fraction of width*

Description

Function to retrieve overlapping intervals that overlap at least by a specified fraction of their widths.

Usage

```
fracOverlap(I1, I2, min.frac=0.0, both=TRUE, mem.friendly=FALSE)
```

Arguments

`I1`	object that inherits from class `Genome_intervals`
`I2`	object that inherits from class `Genome_intervals`
`min.frac`	numeric; minimum required fraction of each of the two interval widths by which two intervals should overlap in order to be marked as overlapping.
`both`	logical; shall both overlap partners meet the minimum fraction `min.frac` requirement? If `FALSE`, then overlaps with only partner involved to at least that fraction are also reported.
`mem.friendly`	logical; if set to `TRUE` an older but memory-friendlier version of `interval_overlap` is used inside this function. Note that `mem.friendly` is only evaluated if `I1` or `I2` is of class `AlignedGenomeIntervals`.

Value

An object of class `data.frame` with one row each for a pair of overlapping elements.

`Index1`	Index of interval in first interval list
`Index2`	Index of interval in second interval list
`n`	number of bases that the two intervals overlap
`fraction1`	fraction of interval 1's width by which the two intervals overlap
`fraction2`	fraction of interval 2's width by which the two intervals overlap

Author(s)

J. Toedling

See Also

`interval_overlap`

Examples

```
data("gen_ints", package="genomeIntervals")
i[4,2] <- 13L
fracOverlap(i, i, 0.5)
```

`getFeatureCounts` *get the read counts for a supplied set of genomic features*

Description

get the read counts for a supplied set of genomic features

Usage

```
getFeatureCounts(AI, FG, nameColumn = "Name", fractionIncluded = 1,
returnType = "AlignedGenomeIntervals", mem.friendly = FALSE)
```

Arguments

`AI`	`AlignedGenomeIntervals` object
`FG`	`Genome\_intervals` objects of genomic features
`nameColumn`	character indicating which column of the object `FG` holds the identifiers of the genomic features; is used to assess the number of genomic copies per feature
`fractionIncluded`	double; which fraction of an interval needs to be included in a feature in order to count for the feature
`returnType`	one of `AlignedGenomeIntervals` or `integer`
`mem.friendly`	logical; passed on to `fracOverlap` function, determines if overlap should be computed chromosome-wise, optionally distributed over several CPUs (with package `parallel`)

Value

depends on argument `returnType`: one of `AlignedGenomeIntervals` or a named `integer`

Author(s)

Joern Toedling

See Also

`fracOverlap`

`girafe-internal` *Internal girafe functions*

Description

Called internally by other `girafe` functions. Normally need not be called by the user.

Author(s)

Wolfgang Huber, Joern Toedling

See Also

`AlignedGenomeIntervals-class`

`intPhred` *Extract integer Phred score values from FastQ data*

Description

Function to extract integer Phred score values from FastQ data.

Usage

```
intPhred(x, method="Sanger", returnType="list")
```

Arguments

`x`	object of class `ShortReadQ`; which contains read sequences and quality scores; usually read in from a Fastq files.
`method`	string; one of 'Sanger', 'Solexa' or 'previousSolexa'. See details below.
`returnType`	string; in which format should the result be returned, either as a 'list' or as a 'matrix'.

Details

There are different standards for encoding read qualities in Fastq files. The 'Sanger' format encodes a Phred quality score from 0 to 93 using ASCII 33 to 126. The current 'Solexa'/Illumina format (1.3 and higher) encodes a Phred quality score from 0 to 40 using ASCII 64 to 104. The 'previous Solexa'/Illumina format (1.0) encodes a custom Solexa/Illumina quality score from -5 to 40 using ASCII 59 to 104. This custom Solexa quality score is approximately equal to the Phred scores for high qualities, but differs in the low quality range.

Value

If `returnType` is equal to 'list': A list of integer Phred quality values of the same length as the number of reads in the object `x`.

If `returnType` is equal to 'matrix': A matrix of integer Phred quality values. The number of rows is the number of reads in the object `x`. The number of columns is the maximum length (width) over all reads in object `x`. The last entries for reads that are shorter than this maximum width are 'NA'.

Author(s)

Joern Toedling

References

`http://maq.sourceforge.net/fastq.shtml`

See Also

`ShortReadQ-class`, `readFastq`

Examples

```
exDir <- system.file("extdata", package="girafe")
ra  <- readFastq(dirPath=exDir, pattern=
          "aravinSRNA_23_plus_adapter_excerpt.fastq")
ra.quals <- intPhred(ra, method="Sanger",
                     returnType="matrix")
ra.qmed  <- apply(ra.quals, 2, median)
if (interactive())
   plot(ra.qmed, type="h", ylim=c(0,42), xlab="Base postion",
        ylab="Median Phred Quality Score", lwd=2, col="steelblue")
```

`medianByPosition` *Compute median quality for each nucleotide position*

Description

This function computes the median quality for each position in a read over all reads in a ShortReadQ object.

Usage

```
medianByPosition(x, method = "Sanger", batchSize = 100000L)
```

Arguments

`x`	object of class `ShortReadQ`, such as the result of function `readFastq`
`method`	string; passed on to function `intPhred`
`batchSize`	number of rows to process in each iteration; directly influences RAM usage of this function

Details

The quality values are computed for each batch of reads and stored as numeric `Rle` objects for each postion. In each iteration, the `Rle` object of the current batch is merged with the previous one in order to keep the RAM usage low.

Value

A numeric vector of the median values per nucleotide position in the reads. The length of this vector corresponds to the length of the longest read in the data.

Author(s)

Joern Toedling

See Also

`intPhred`

Examples

```
exDir <- system.file("extdata", package="girafe")
ra  <- readFastq(dirPath=exDir, pattern=
                 "aravinSRNA_23_plus_adapter_excerpt.fastq")
medianByPosition(ra, batchSize=200)
```

`perWindow` *Investigate aligned reads in genome intervals with sliding windows*

Description

Investigate aligned reads in genome intervals with sliding windows.

Usage

```
perWindow(object, chr, winsize, step, normaliseByMatches = TRUE,
          mem.friendly = FALSE)
```

Arguments

`object`	object of class `AlignedGenomeIntervals`
`chr`	string; which chromosome to investigate with sliding windows
`winsize`	integer; size of the sliding window in base-pairs
`step`	integer; offset between the start positions of two sliding windows
`normaliseByMatches`	logical; should the number of reads per `AlignedGenomeInterval` be normalised by the number of genomic matches of the read sequence before summing them up in each window? (*i.e.* derivation a weighted sum of read counts)
`mem.friendly`	logical; argument passed on to function `interval_overlap`; if `TRUE` the less RAM and, if the `parallel` package is attached, multiple processors are used for computing the overlap, on the expense of time

Details

The windows are constructed from the first base position onto which a read has been mapped until the end of the chromosome.

Value

a `data.frame` with the following information for each sliding window on the chromosome

`chr`	string; which chromosome the interval is on
`start`	integer; start coordinate of the windows on the chromosome
`end`	integer; end coordinate of the windows on the chromosome
`n.overlap`	integer; number of read match positions inside the window. Per match position there can be one or more reads mapped, so this number always is smaller than `n.reads`
`n.reads`	numeric; number of reads which match positions inside this window; can be floating-point numbers if argument `normaliseByMatches=TRUE`

`n.unique`	integer; number of reads which each only have one match position in the genome and for which this position is contained inside this window
`max.reads`	integer; the maximal number of reads at any single one match position contained inside this window
`first`	integer; coordinate of the first read alignment found inside the window
`last`	integer; coordinate of the last read alignment found inside the window

The result is of class `data.frame` and in addition of the (S3) class `slidingWindowSummary`, which may be utilized by follow-up functions.

Author(s)

Joern Toedling

See Also

`AlignedGenomeIntervals-class`

Examples

```
exDir <- system.file("extdata", package="girafe")
exA   <- readAligned(dirPath=exDir, type="Bowtie",
  pattern="aravinSRNA_23_no_adapter_excerpt_mm9_unmasked.bwtmap")
exAI  <- as(exA, "AlignedGenomeIntervals")
exPX  <- perWindow(exAI, chr="chrX", winsize=1e5, step=0.5e5)
head(exPX[order(exPX$n.overlap, decreasing=TRUE),])
```

`plotAligned` *Visualise reads aligned to genome intervals*

Description

Visualise reads aligned to genome intervals

Usage

```
plotAligned(x, y, chr, start, end, plus.col = "#00441b",
   minus.col = "#283d78", gff, featureLegend = FALSE,
   gffChrColumn = "seq_name", gffTypeColumn="type",
   gffNameColumn="ID",
   featureExclude = c("chromosome", "nucleotide_match", "insertion"),
   showStrands="both", extraColors=NULL, ylim, highlight, main, ...)
```

Arguments

`x`	Object of class `AlignedGenomeIntervals`
`y`	This argument is only specified for compatibility with plot.default and not used in the function.
`chr`	string; on which chromosome is the region to plot
`start`	integer; start coordinate of the chromosome region to plot
`end`	integer; end coordinate of the chromosome region to plot

`plus.col`	which colour to use for the reads on the Plus strand
`minus.col`	which colour to use for the reads on the Plus strand
`gff`	Data frame containing annotation for genomic feature to be used to further annotate the plot. Note that it must include a column called “type” that indicates the type of each genomic feature (e.g. miRNA, gene etc.).
`featureLegend`	logical; should a legend that describes the colour code for the annotated genome features be appended at the bottom of the plot?
`gffChrColumn`	string; which column of the `gff` data.frame holds the chromosome identifier of each feature.
`gffTypeColumn`	string; which column of the `gff` data.frame holds the type/class identifier of each feature. Used for the colouring of features.
`gffNameColumn`	what is the column of the gff data.frame called that holds the identifier of the element that should be displayed in the plot; default: “name”
`featureExclude`	character; which kinds of annotated genome features specified in the `gff` are to be ignored for the plot
`showStrands`	string; which strands to show in the plot; defaults to “both”, but users can specify to show only the reads on “plus” or “minus” strand
`extraColors`	named character vector which allows the user to specify custom colours for feature types; colours must be specified in RGB format as hexadecimal strings starting with “#”, e.g. “#addfff” for light-blue
`ylim`	range of read numbers to plot (y-axis limits); if not specified they are computed from the data in the specified region
`highlight`	currently unused
`main`	string; main title to use for the plot
`...`	further arguments passed on to the more primitive plotting functions used

Details

This function implements the `plot` method for objects of class `AlignedGenomeIntervals`.

Value

Returns `NULL`; this function is called for the side-effect of creating the plot.

Note

This function was inspired by and borrows source code from the function `plotAlongChrom` in package `tilingArray`

Author(s)

Joern Toedling, Wolfgang Huber

See Also

`AlignedGenomeIntervals-class`

Examples

```
A <- AlignedGenomeIntervals(
      start=c(1,8,10,20), end=c(5,15,16,25),
      chromosome=rep("chr1", each=4),
      strand=c("+","+","+","+"),
      sequence=c("ACATT","TATCGGAC","TCGGACT","GTAACG"),
      reads=c(5L, 2L, 4L, 7L) )
M2 <- new("Genome_intervals_stranded",
          rbind(c(2,6), c(1,15), c(20,30)),
          closed = matrix(TRUE, ncol=2, nrow=3),
          annotation = data.frame(
            seq_name= factor(rep("chr1", 3)),
            inter_base= logical(3),
            strand=factor(rep("+", 3), levels=c("+","-")),
            alias=c("miRNA1","gene1","tRNA1"),
            type=c("miRNA","gene","tRNA")) )
if (interactive())
  plot(A, M2, chr="chr1", start=0, end=35,
       nameColum="alias", showStrands="plus")

if (interactive())
## use 'extraColors' to replace default colours (or to add new ones):
  plot(A, M2, chr="chr1", start=0, end=35, nameColum="alias",
       showStrands="plus", extraColors=c("tRNA"="#6666DD"))

## See also the examples in the vignette and on the manual page
##  of the class 'AlignedGenomeIntervals'
```

`plotNegBinomFit` *Plot Negative Binomial Fit*

Description

Plot Negative Binomial Fit

Usage

```
plotNegBinomFit(x, breaks = c(-0.5:15.5, 1e+08), bar.col=rainbow(2),
                 addLegend=TRUE, legend.names=c("data","background"), ...)
```

Arguments

`x`	data.frame; slidingWindowSummary
`breaks`	numeric vector of breakpoints
`bar.col`	colours for the bars
`addLegend`	logical; should a legend be added in the top-right corner of the plot
`legend.names`	character; names for the legend
`...`	further arguments passed on to function barplot

Value

returns NULL; only called for its side-effect of producing the plot

Author(s)

J Toedling

See Also

`barplot`

`plotReads` *Function to plot aligned reads along the chromosome*

Description

Function to plot aligned reads along the chromosome

Usage

```
plotReads(dat, ylim, strand = "plus", vpr, sampleColor = NULL,
   zeroLine = FALSE, main, pointSize = unit(1, "mm"),
   cexAxisLabel = 1, cexAxis = 1, ylab, ...)
```

Arguments

`dat`	a list with arguments **x.start** integer; the genome start coordinates of the data to visualise **x.end** integer; the genome end coordinates of the data to visualise **y** numeric; the levels of the data to visualise **flag** numeric; specifies the category of each value, e.g. marks which data values belong to unique read alignments and which not
`ylim`	y-axis limits of the plotting window
`strand`	string; which of the two strands is plotted
`vpr`	which viewport to use for this plot
`sampleColor`	which colour to use for the data
`zeroLine`	logical; should a line at y=0 be drawn?
`main`	string; main title for the plot
`pointSize`	width of each dot/bar
`cexAxisLabel`	numeric; expansion factor for the axis labels
`cexAxis`	numeric; expansion factor for the axis labels
`ylab`	y-axis label
`...`	further arguments passed on to the more primitive plotting functions that are used

Details

This function is used inside the plotting method for objects of class `AlignedGenomeIntervals`.

Value

returns `Null`; called for plotting single reads inside the function `plotAligned`

Author(s)

Joern Toedling

`trimAdapter` *Remove 3' adapter contamination*

Description

Function to remove 3' adapter contamination from reads

Usage

```
trimAdapter(fq, adapter, match.score = 1, mismatch.score = -1,
            score.threshold = 2)
```

Arguments

`fq`	Object of class `ShortReadQ`; the reads with possible adapter contamination.
`adapter`	object of class `DNAString` or class `character`; the sequence of the 3' adapter which could give rise to the 3' contamination. If of class `character`, it is converted to a `DNAString` inside the function.
`match.score`	numeric; alignment score for matching bases
`mismatch.score`	numeric; alignment score for mismatches
`score.threshold`	numeric; minimum total alignment score required for an overlap match between the 3' end of the read and the 5' end of the adapter sequence.

Details

Performs an overlap alignment between the ends of the reads and the start of the adapter sequence.

Value

An object of class `ShortReadQ` containing the reads without the 3' adapter contamination.

Note

The function `trimLRPatterns` from package `ShortRead` may be a faster alternative to this function.

Author(s)

J. Toedling

See Also

`pairwiseAlignment`, `narrow`, `readFastq`, `writeFastq`

Examples

```
exDir <- system.file("extdata", package="girafe")
## load reads containing adapter fragments at the end
ra23.wa  <- readFastq(dirPath=exDir, pattern=
            "aravinSRNA_23_plus_adapter_excerpt.fastq")
table(width(ra23.wa))
# adapter sequence obtained from GEO page
#  accession number: GSE10364
#adapter <- DNAString("CTGTAGGCACCATCAAT")
adapter <- "CTGTAGGCACCATCAAT"

# trim adapter
ra23.na  <- trimAdapter(ra23.wa, adapter)
table(width(ra23.na))
```

`weightedConsensusMatrix`

compute weighted consensus matrix

Description

computes weighted consensus matrix

Usage

```
weightedConsensusMatrix(seqs, weights, shift = NULL,
                           baseLetters = c("A", "C", "G", "T", "N"))
```

Arguments

`seqs`	character vector of read sequences
`weights`	integer; weights (read counts)
`shift`	integer; shift of each read sequence relative to the first column of the consensus matrix, by default: 0
`baseLetters`	alphabet

Value

A consensus matrix

Author(s)

J Toedling

See Also

`consensusMatrix`

Examples

```
## Align following sequences with weights:
##   ACATT       1
##    CGTTA     10
##     TTG       3
##  GACATT       4

dweights <- c(1L, 10L, 3L, 4L)
d <- c("ACATT","CGTTA", "TTG", "GACATT")
dshifts <- c(0L, 1L, 2L, -1L)
W <- girafe:::weightedConsensusMatrix(d, dweights, shift=dshifts)
consensusString(W, ambiguityMap="N")
```

`which_nearest-methods` *Methods for function 'which_nearest' and genome intervals*

Description

For each genome interval in one set, finds the nearest interval in a second set of genome intervals.

Value

a `data.frame` with a number of rows equal to the number of intervals in argument `from`. The elements of the data.frame are:

`distance_to_nearest`	numeric; distance to nearest interval from object `to`. Is 0 if the current interval in object `from` did overlap one or more intervals in object `to`
`which_nearest`	list; each list element are the indices or the intervals in object `to` that have the closest distance to the current interval in object `from`
`which_overlap`	list; each list element are the indices or the intervals in object `to` that do overlap with the current interval in object `from`

Methods

Currently, the package *girafe* contains method implementations for the first object (Argument: `from`) being of any of the classes “AlignedGenomeIntervals”,“Genome_intervals” or “Genome_intervals_stranded”. The second object (Argument: `to`) has be of class “Genome_intervals_stranded” or “Genome_intervals”.

Note

If the supplied objects are stranded, as it is the case with objects of classes ‘AlignedGenomeIntervals’ and ‘Genome_intervals_stranded’, then the overlap and distance is solely computed between intervals on the same strand.

For objects of class ‘Genome_intervals’, overlap and distances are computed regardless of strand information.

Author(s)

Joern Toedling

See Also

which_nearest

Examples

```
### process aligned reads
exDir <- system.file("extdata", package="girafe")
exA   <- readAligned(dirPath=exDir, type="Bowtie",
  pattern="aravinSRNA_23_no_adapter_excerpt_mm9_unmasked.bwtmap")
exAI <- as(exA, "AlignedGenomeIntervals")

## load annotated genome features
load(file.path(exDir, "mgi_gi.RData"))

## subset for sake of speed:
A <- exAI[is.element(seqnames(exAI), c("chrX","chrY"))]
G <- mgi.gi[is.element(seqnames(mgi.gi), c("chrX","chrY"))]

## find nearest annotated feature for each AlignedGenomeInterval
WN <- which_nearest(A, G)
dim(WN); tail(WN)

## notice the difference to:
tail(which_nearest(as(A, "Genome_intervals"), G))
# the last interval in A is located antisense to a gene,
# but not overlapping anything on the same strand
```

Index