girafe

October 5, 2010

`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 utilized 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

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`

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

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 seperately for each strand. If `byStrand=FALSE` (default) only one result is returned per chromosome. If `byStrand=TRUE`, each chromosome result is again a list with two separate `Rle` objects.
By now, the `coverage` method for `AlignedGenomeIntervals` makes use of the method for `RangesList` objects from package `IRanges` (thanks to a suggestion from P. Aboyoun).

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 greather than or equal to 0. They specifiy 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.

interval_overlap determine the overlap between the intervals in an object of class `AlignedGenomeIntervals` and other intervals that may be supplied as an object of one of the classes `Genome_intervals`, `Genome_intervals_stranded`, or `AlignedGenomeIntervals`.

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 visualization 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. Intervals are only combined if they are on the same chromosome, the same strand

AND have the same 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 optional argument `exact=TRUE`.

score access or set a custom score for the 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`, `RangedData-class`, `RangedData-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(1L, 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("chr1","chr2","chr3"), each=2),
   strand=c("-","-","+","+","+","+"),
   sequence=c("ACATT","ACA","CGT","GTAA","AG","CT"),
   reads=rep(1 ,6))
detail(A)

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

## the 'reduce' method provides a cleaned-up, compact set
detail(A)
detail(reduce(A))

## 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, "anno_mm_genint.RData"))
plot(exAI, mm.gi, chr="chrX", start=50400000, end=50410000)

### overlap with annotated genome elements:
exOv <- interval_overlap(exAI, mm.gi)
## how many elements do read match positions generally overlap:
table(listLen(exOv))
## what are the 12 elements overlapped by a single match position:
mm.gi[exOv[[which(listLen(exOv)==12)]]]
## what kinds of elements are overlapped
(tabOv <- table(as.character(mm.gi$type)[unlist(exOv)]))
### display those classes:
my.cols <- rainbow(length(tabOv))
pie(tabOv, col=my.cols, radius=0.95)
```

`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 = 0.66)
```

Arguments

`I1`	object that inherits from class `Genome_intervals`
`I2`	object that inherits from class `Genome_intervals`
`min.frac`	numeric; minimum fraction of smaller of the two interval widths by which two intervals should overlap in order to be marked as overlapping.

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
`fraction`	fraction of the smaller of the two intervals' width by which the two intervals overlap
`frac1`	fraction of interval 1's width by which the two intervals overlap
`frac2`	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] <- 13
fracOverlap(i, i)
```

`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'/llumina 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)
plot(ra.qmed, type="h", ylim=c(0,42), xlab="Base postion",
     ylab="Median Phred Quality Score", lwd=2, col="steelblue")
```

`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 knowns 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,]
```

`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, organism = "Mm")
```

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
`organism`	string; with the data of which organism are we dealing here

Details

The windows are constructed from the first base position onto which a read has been mapped until the end of the chromosome. Chromosome lengths are obtained from organism annotation packages, such as `org.mm.eg.db`.

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`	integer; number of reads which match positions inside this window
`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

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),])
```

`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))
```

`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, "anno_mm_genint.RData"))

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

∗Topic **classes**
    AlignedGenomeIntervals-class, 1
∗Topic **manip**
    addNBSignificance, 6
    fracOverlap, 4
    intPhred, 5
    perWindow, 8
    trimAdapter, 9
∗Topic **methods**
    which_nearest-methods, 10
[,AlignedGenomeIntervals,ANY,ANY-method *(AlignedGenomeIntervals-class)*, 1

addNBSignificance, 6
AlignedGenomeIntervals *(AlignedGenomeIntervals-class)*, 1
AlignedGenomeIntervals-class, *9*
AlignedGenomeIntervals-class, 1
AlignedRead-class, *3*

c,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
c.AlignedGenomeIntervals *(AlignedGenomeIntervals-class)*, 1
chromosome,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
chromosome,Genome_intervals-method *(AlignedGenomeIntervals-class)*, 1
clusters,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
clusters,Genome_intervals-method *(AlignedGenomeIntervals-class)*, 1
coerce,AlignedGenomeIntervals,RangedData-method *(AlignedGenomeIntervals-class)*, 1
coerce,AlignedRead,AlignedGenomeIntervals-meth *(AlignedGenomeIntervals-class)*, 1
coverage,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1

detail,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1

estimateNBParams *(addNBSignificance)*, 6
export *(AlignedGenomeIntervals-class)*, 1
export,AlignedGenomeIntervals,character,charac *(AlignedGenomeIntervals-class)*, 1
extend *(AlignedGenomeIntervals-class)*, 1
extend,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
extend,Genome_intervals-method *(AlignedGenomeIntervals-class)*, 1
extend,Genome_intervals_stranded-method *(AlignedGenomeIntervals-class)*, 1

fracOverlap, 4

Genome_intervals-class, *2, 3*

interval_included,AlignedGenomeIntervals,Align *(AlignedGenomeIntervals-class)*, 1
interval_included,AlignedGenomeIntervals,Geno *(AlignedGenomeIntervals-class)*, 1
interval_included,Genome_intervals_stranded,Al *(AlignedGenomeIntervals-class)*, 1

interval_overlap, *5*
interval_overlap,AlignedGenomeIntervals,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
interval_overlap,AlignedGenomeIntervals,Genome_intervals-method *(AlignedGenomeIntervals-class)*, 1
interval_overlap,AlignedGenomeIntervals,Genome_intervals_stranded-method *(AlignedGenomeIntervals-class)*, 1
interval_overlap,Genome_intervals,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
interval_overlap,Genome_intervals_stranded,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
Intervals_full, *2*
intPhred, 5

matches *(AlignedGenomeIntervals-class)*, 1
matches,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
matches<- *(AlignedGenomeIntervals-class)*, 1
matches<-,AlignedGenomeIntervals,integer-method *(AlignedGenomeIntervals-class)*, 1

narrow, *10*

organism,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
organism<- *(AlignedGenomeIntervals-class)*, 1
organism<-,AlignedGenomeIntervals,character-method *(AlignedGenomeIntervals-class)*, 1

p.adjust, *7*
pairwiseAlignment, *10*
perWindow, *6, 7*, 8
plot,AlignedGenomeIntervals,ANY-method *(AlignedGenomeIntervals-class)*, 1
plot,AlignedGenomeIntervals,Genome_intervals_stranded-method *(AlignedGenomeIntervals-class)*, 1
plot,AlignedGenomeIntervals,missing-method *(AlignedGenomeIntervals-class)*, 1
plot,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
plotAligned, *2, 3*

RangedData-class, *3*
readFastq, *6, 10*
reads *(AlignedGenomeIntervals-class)*, 1
reads,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
reads<- *(AlignedGenomeIntervals-class)*, 1
reads<-,AlignedGenomeIntervals,character-metho *(AlignedGenomeIntervals-class)*, 1
reduce,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
reduce,Genome_intervals-method *(AlignedGenomeIntervals-class)*, 1

score,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
score<- *(AlignedGenomeIntervals-class)*, 1
score<-,AlignedGenomeIntervals,numeric-method *(AlignedGenomeIntervals-class)*, 1
seq_name *(AlignedGenomeIntervals-class)*, 1
seq_name,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
seq_name,Genome_intervals-method *(AlignedGenomeIntervals-class)*, 1
ShortReadQ-class, *6*
show,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*, 1
strand,AlignedGenomeIntervals-method *(AlignedGenomeIntervals-class)*,

1
`strand<-,AlignedGenomeIntervals,factor-method` *(`AlignedGenomeIntervals-class`)*, 1
`strand<-,AlignedGenomeIntervals,vector-method` *(`AlignedGenomeIntervals-class`)*, 1
`subset,AlignedGenomeIntervals-method` *(`AlignedGenomeIntervals-class`)*, 1

`trimAdapter`, 9

`which_nearest`, *11*
`which_nearest` *(`which_nearest-methods`)*, 10
`which_nearest,AlignedGenomeIntervals,Genome_intervals_stranded-method` *(`which_nearest-methods`)*, 10
`which_nearest,Genome_intervals,Genome_intervals-method` *(`which_nearest-methods`)*, 10
`which_nearest,Genome_intervals_stranded,Genome_intervals_stranded-method` *(`which_nearest-methods`)*, 10
`which_nearest-methods`, 10
`width,AlignedGenomeIntervals-method` *(`AlignedGenomeIntervals-class`)*, 1
`writeFastq`, *10*