Package ‘bamsignals’

October 17, 2017

Type Package

Title Extract read count signals from bam files

Version 1.8.0

Date 2016-03-30

Author Alessandro Mammana [aut, cre], Johannes Helmuth [aut]

Maintainer Alessandro Mammana <mammana@molgen.mpg.de>

Description This package allows to efficiently obtain count vectors from indexed bam files. It counts the number of reads in given genomic ranges and it computes reads profiles and coverage profiles. It also handles paired-end data.

License GPL-2

Depends R (>= 3.2.0)

Imports methods, BiocGenerics, Rcpp (>= 0.10.6), IRanges, GenomicRanges, zlibbioc

Suggests testthat (>= 0.9), Rsamtools, BiocStyle, knitr, rmarkdown

LinkingTo Rcpp, Rhtslib, zlibbioc

biocViews DataImport, Sequencing, Coverage, Alignment

VignetteBuilder knitr

NeedsCompilation yes

RoxygenNote 5.0.1

R topics documented:

`as.list.CountSignals` *Converts the container to a list `l` such that `l[[i]]` is the i-th signal.*

Description

Converts the container to a list `l` such that `l[[i]]` is the i-th signal.

Usage

```
## S3 method for class 'CountSignals'
as.list(x, ...)
```

Arguments

`x`	A CountSignals object
`...`	not used

Value

a list `l` such that `l[[i]]` is `x[i]`.

`bamsignals` *Efficient counting of reads in a bam file*

Description

Functions and classes for extracting signals from a bam file. Differently than the other packages, this package is not supposed to import reads in R. All the read-processing is done in C/C++ and the only output are read counts.

Author(s)

Alessandro Mammana <`mammana@molgen.mpg.de`>

Johannes Helmuth <`helmuth@molgen.mpg.de`>

See Also

`bamsignals-methods` for the functions and `CountSignals` for the class.

`bamsignals-methods` *Functions for extracting count signals from a bam file.*

Description

Functions for extracting count signals from a bam file.

`bamCount`: for each range, count the reads whose 5' end map in it.

`bamProfile`: for each base pair in the ranges, compute the number of reads whose 5' end maps there.

`bamCoverage`: for each base pair in the ranges, compute the number of reads covering it.

Usage

```
## S4 method for signature 'character,GenomicRanges'
bamCount(bampath, gr, mapqual = 0,
  shift = 0, ss = FALSE, paired.end = c("ignore", "filter", "midpoint"),
  tlenFilter = NULL, filteredFlag = -1, verbose = TRUE)

## S4 method for signature 'character,GenomicRanges'
bamProfile(bampath, gr, binsize = 1,
  mapqual = 0, shift = 0, ss = FALSE, paired.end = c("ignore", "filter",
  "midpoint"), tlenFilter = NULL, filteredFlag = -1, verbose = TRUE)

## S4 method for signature 'character,GenomicRanges'
bamCoverage(bampath, gr, mapqual = 0,
  paired.end = c("ignore", "extend"), tlenFilter = NULL,
  filteredFlag = -1, verbose = TRUE)
```

Arguments

`bampath`	path to the bam file storing the read. The file must be indexed.
`gr`	GenomicRanges object used to specify the regions. If a range is on the negative strand the profile will be reverse-complemented.
`mapqual`	discard reads with mapping quality strictly lower than this parameter. The value 0 ensures that no read will be discarded, the value 254 that only reads with the highest possible mapping quality will be considered.
`shift`	shift the read position by a user-defined number of basepairs. This can be handy in the analysis of chip-seq data.
`ss`	produce a strand-specific profile or ignore the strand of the read. This option does not have any effect on the strand of the region. Strand-specific profiles are twice as long then strand-independent profiles.
`paired.end`	a character string indicating how to handle paired-end reads. If `paired.end!="ignore"` then only first reads in proper mapped pairs will be considered (SAMFLAG 66, i.e. in the flag of the read, the bits in the mask 66 must be all ones). If `paired.end=="midpoint"` then the midpoint of a filtered fragment is considered, where `mid = fragment_start + int(abs(tlen)/2)`, and where tlen is the template length stored in the bam file. For even tlen, the given midpoint will be moved of 0.5 basepairs in the 3' direction (bamCount, bamProfile). If `paired.end=="extend"` then the whole fragment is treated as a single read (bamCoverage).

`tlenFilter`	A filter on fragment length as estimated from alignment in paired end experiments (TLEN). If set to `c(min,max)` only reads are considered where `min <= TLEN <= max`. If `paired.end=="ignore"`, this argument is set to `NULL` and no filtering is done. If `paired.end!="ignore"`, this argument defaults to `c(0,1000)`.
`filteredFlag`	Filter out reads with a certain flag set, e.g. "1024" to filter out PCR or optical duplicates.
`verbose`	a logical value indicating whether verbose output is desired
`binsize`	If the value is set to 1, the method will return basepair-resolution read densities, for bigger values the density profiles will be binned (and the memory requirements will scale accordingly).

Details

A read position is always specified by its 5' end, so a read mapping to the reference strand is positioned at its leftmost coordinate, a read mapping to the alternative strand is positioned at its rightmost coordinate. This can be changed using the `shift` parameter.

Value

- `bamProfile` and `bamCoverage`: a CountSignals object with a signal per region
- `bamCount`: a vector with one element per region or, if `ss==TRUE`, a matrix with one column per region and two rows (sense and antisense).

See Also

`CountSignals` for handling the return value of `bamProfile` and `bamCoverage`

Examples

```
## TOY DATA ##
library(GenomicRanges)
bampath <-
system.file("extdata", "randomBam.bam", package="bamsignals")
genes <-
get(load(system.file("extdata", "randomAnnot.Rdata", package="bamsignals")))


## THE FUNCTION 'count' ##
#count how many reads map in each region (according to 5' end)
v <- bamCount(bampath, genes)
#plot it
labs <- paste0(seqnames(genes), ":", start(genes), "-", end(genes))
par(mar=c(5, 6, 4, 2))
barplot(v, names.arg=labs, main="read counts per region", las=2,
horiz=TRUE, cex.names=.6)

#distinguish between strands
v2 <- bamCount(bampath, genes, ss=TRUE)
#plot it
par(mar=c(5, 6, 4, 2))
barplot(v2, names.arg=labs, main="read counts per region", las=2,
horiz=TRUE, cex.names=.6, col=c("blue", "red"), legend=TRUE)
```

```
## THE FUNCTIONS 'bamProfile' and 'bamCoverage' ##
#count how many reads map to each base pair (according to 5' end)
pu <- bamProfile(bampath, genes)
#count how many reads cover each base pair
du <- bamCoverage(bampath, genes)
#plot it
xlab <- "offset from start of the region"
ylab <- "reads per base pair"
main <- paste0("read coverage and profile of the region ", labs[1])
plot(du[1], ylim=c(0, max(du[1])), ylab=ylab, xlab=xlab, main=main, type="l")
lines(pu[1], lty=2)
llab <- c("covering the base pair", "5' end maps to the base pair")
legend("topright", llab, lty=c(1,2), bg="white")



## REGIONS OF THE SAME SIZE AND OPTIONS FOR 'bamProfile' ##
proms <- promoters(genes, upstream=150, downstream=150)
#pileup according to strand
pu_ss <- bamProfile(bampath, proms, ss=TRUE)
#compute average over regions
avg_ss <- apply(alignSignals(pu_ss), 2, rowMeans)

#profile using a strand-specific shift
pu_shift <- bamProfile(bampath, proms, shift=75)
#compute average over regions
avg_shift <- rowMeans(alignSignals(pu_shift))

#profile using a strand-specific shift and a binning scheme
binsize <- 20
pu_shift_bin <- bamProfile(bampath, proms, shift=75, binsize=binsize)
#compute average over regions
avg_shift_bin <- rowMeans(alignSignals(pu_shift_bin))

#plot it
xs <- -149:150
main <- paste0("average read profile over ", length(proms), " promoters")
xlab <- "distance from TSS"
ylab <- "average reads per base pair"
plot(xs, avg_shift, xlab=xlab, ylab=ylab, main=main, type="l",
ylim=c(0, max(avg_shift)))
lines(xs, avg_ss["sense",], col="blue")
lines(xs, avg_ss["antisense",], col="red")
lines(xs, rep(avg_shift_bin/binsize, each=binsize), lty=2)
llabs <-
c("sense reads", "antisense reads", "with shift", "binned and with shift")
legend("topright", llabs, col=c("blue", "red", "black", "black"),
lty=c(1,1,1,2), bg="white")
```

`CountSignals-class` *Container for count signals*

Description

This s4 class is a tiny wrapper around a normal list (stored in the `signals` slot) and it is the output of the methods in the bamsignals package. Among other things the container provides an accessor method, that returns single signals as vectors and matrices, and the methods `as.list` and `alignSignals`, that convert the container to a list or an array/matrix respectively. A CountSignals object is read-only, i.e. it cannot be modified.

Usage

```
## S4 method for signature 'CountSignals'
length(x)

## S4 method for signature 'CountSignals'
width(x)

## S4 method for signature 'CountSignals,ANY'
x[i, drop = TRUE]

## S4 method for signature 'CountSignals'
as.list(x)

## S4 method for signature 'CountSignals'
alignSignals(x)
```

Arguments

`x`	A CountSignals object
`i`	Index for subsetting. It can be a single index as well as a vector of indices.
`drop`	In case `i` is a vector of length 1, after subsetting, collapse the CountSignal object to a single signal or not.

Value

return values are described in the Methods section.

Methods (by generic)

- `length`: Number of contained signals
- `width`: Width of each signal. If the CountSignals object `csig` is strand-specific then `width(csig)[i] == ncol(csi` otherwise `width(csig)[i] = length(csig[i])`.
- `[`: Access single signals or subset the CountSignals object. If `i` is a single index and `drop==TRUE` then the accessor returns a single signal. If `x` is strand-specific then a single signal is a matrix with two rows, the first for the sense, the second for the antisense strand. Otherwise a signle signal is simply a vector of integers. If `i` is a vector of length different than 1, then the acessor returns a subset of the CountSignals object. Invalid indices result into errors.
- `as.list`: Converts the container to a list `l` such that `l[[i]]` is the i-th signal.
- `alignSignals`: Convert to a matrix or to an array. This is only possible if all signals have the same width `w`. If the CountSignals object `csig` is strand-specific, the result is an array of dimensions `[2, w, length(csig)]`, otherwise it will be a matrix of dimensions `[w, length(csig)]`.

Slots

- `ss` A single boolean value indicating whether all signals are strand-specific or not
- `signals` A list of integer vectors (if `ss==TRUE`) or of integer matrices, representing each signal

See Also

`bamsignals-methods` for the functions that produce this object

Examples

```
#get a CountSignals object
library(GenomicRanges)
bampath <-
system.file("extdata", "randomBam.bam", package="bamsignals")
genes <-
get(load(system.file("extdata", "randomAnnot.Rdata", package="bamsignals")))
csig <- bamProfile(bampath, genes, ss=TRUE)

#show it
show(csig)

#number of contained signals
len <- length(csig)

#width of each signal
w <- width(csig)

#get one element as a vector (or matrix)
v <- csig[1]

#use as if it was a list
tot_per_sig <- sapply(csig, sum)

#convert to a list
siglist <- as.list(csig)

#get regions and signals of the same width
proms <- promoters(genes, upstream=150, downstream=150)
csig <- bamCoverage(bampath, proms)

#convert to matrix
mat <- alignSignals(csig)
```

Index