Package 'csaw'

April 9, 2015

Version 1.0.7

Date 2014/03/11

Title ChIP-seq analysis with windows

Author Aaron Lun <alun@wehi.edu.au>, Gordon Smyth <smyth@wehi.edu.au>

Maintainer Aaron Lun <alun@wehi.edu.au>

Depends R (>= 3.1.0), GenomicRanges

Imports Rsamtools, edgeR, limma, GenomicFeatures, AnnotationDbi, methods, GenomicAlignments, S4Vectors, IRanges, GenomeInfoDb

Suggests org.Mm.eg.db, TxDb.Mmusculus.UCSC.mm10.knownGene

biocViews MultipleComparison, ChIPSeq, Normalization, Sequencing, Coverage, Genetics, Annotation

Description Detection of differentially bound regions in ChIP-seq data with sliding windows, with methods for normalization and proper FDR control.

License GPL-3

R topics documented:

`combineTests` *Combine statistics across multiple tests*

Description

Combines p-values across clustered tests using Simes' method to control the cluster FDR.

Usage

```
combineTests(ids, tab, weight=rep(1, length(ids)))
```

Arguments

`ids`	an integer vector containing the cluster ID for each test
`tab`	a dataframe of results with `PValue`, `logCPM` and at least one `logFC` field for each test
`weight`	a numeric vector of weights for each window

Details

This function uses Simes' procedure to compute the combined p-value for each cluster of tests. Each p-value represents evidence against the global null hypothesis, i.e., all individual nulls are true in each cluster. This may be more relevant than examining each test individually when multiple tests in a cluster represent parts of the same underlying event, e.g., genomic regions consisting of clusters of windows.

Mean logFC and logCPM values are also computed across all tests in each cluster. Multiple fields in `tab` containing the `logFC` substring are allowed, e.g., to accommodate ANOVA-like contrasts. Note that the average may not be a suitably informative metric when clusters are large and heterogenous, in which case custom summaries may be required. The BH method is also applied to control the FDR across all clusters.

The importance of each test within a cluster can be adjusted by supplying different relative `weight` values. This may be useful for downweighting low-confidence tests, e.g., those in repeat regions. In Simes' procedure, weights are interpreted as relative frequencies of the tests in each cluster. Note that these weights have no effect between clusters and will not be used to adjust the computed FDR.

A simple clustering approach for windows is provided in `mergeWindows`. However, anything can be used so long as it does not compromise type I error control, e.g., promoters, gene bodies, independently called peaks.

Value

A dataframe with one row per cluster and the numeric fields `logCPM`, the average log-CPM; `PValue`, the combined p-value; and `FDR`, the q-value corresponding to the combined p-value. There is also one numeric field representing the average of each supplied `logFC` in `tab`. The name of each row corresponds to the sorted cluster IDs.

Author(s)

Aaron Lun

References

Simes RJ (1986). An improved Bonferroni procedure for multiple tests of significance. *Biometrika* 73, 751-754.

Benjamini Y and Hochberg Y (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. R. Stat. Soc. Series B* 57, 289-300.

Lun ATL and Smyth GK (2014). De novo detection of differentially bound regions for ChIP-seq data using peaks and windows: controlling error rates correctly. *Nucleic Acids Res.* 42, e95

See Also

mergeWindows

Examples

```
ids <- round(runif(100, 1, 10))
tab <- data.frame(logFC=rnorm(100), logCPM=rnorm(100),
    PValue=rbeta(100, 1, 2))
combined <- combineTests(ids, tab)
head(combined)

# With window weighting.
w <- round(runif(100, 1, 5))
combined <- combineTests(ids, tab, weight=w)
head(combined)

# With multiple log-FCs.
tab$logFC.whee <- rnorm(100, 5)
combined <- combineTests(ids, tab)
head(combined)
```

---

correlateReads *Compute correlation coefficients between reads*

Description

Computes the auto- or cross-correlation coefficients between read positions across a set of delay intervals.

Usage

```
correlateReads(bam.files, max.dist=1000, cross=TRUE, param=readParam())
```

Arguments

bam.files	a character vector containing paths to sorted and indexed BAM files
max.dist	integer scalar specifying the maximum delay distance over which correlation coefficients will be calculated
cross	a logical scalar specifying whether cross-correlations should be computed
param	a readParam object containing read extraction parameters

Details

If cross=TRUE, reads are separated into those mapping on the forward and reverse strands. Positions on the forward strand are shifted forward by a delay interval. The chromosome-wide correlation coefficient between the shifted forward positions and the original reverse positions are computed. This is repeated for all delay intervals less than maxDist. A weighted mean for the cross-correlation is taken across all chromosomes, with weighting based on the number of reads.

Cross-correlation plots can be used to check the quality of immunoprecipitation for ChIP-Seq experiments involving transcription factors or punctate histone marks. Strong immunoprecipitation should result in a peak at a delay corresponding to the fragment length. A spike may also be observed at the delay corresponding to the read length. This is probably an artefact of the mapping process where unique mapping occurs to the same sequence on each strand.

The construction of cross-correlation plots is usually uninformative for full paired-end data. This is because the presence of valid pairs will inevitably result in a strong peak at the fragment length. Nonetheless, immunoprecipitation efficiency can be diagnosed by treating paired-end data as single end data. This is done by using only the first or second read based on the value of pet used in readParam. Setting pet="both" will result in failure.

If multiple BAM files are specified in bam.files, the reads from all libraries are pooled prior to calculation of the correlation coefficients. This is convenient for determining the average correlation profile across an entire dataset. Separate calculations for each file will require multiple calls to correlateReads.

If cross=FALSE, auto-correlation coefficients are computed without use of strand information. This is designed to guide estimation of the average width of enrichment for diffuse histone marks. For example, the width can be defined as the delay distance at which the autocorrelations become negligble. However, this tends to be ineffective in practice as diffuse marks tend to have very weak correlations to begin with.

By default, marked duplicate reads are removed from each BAM file prior to calculation of coefficients. This is strongly recommended, even if the rest of the analysis will be performed with duplicates retained. Otherwise, the read length spike will dominate the plot. The fragment length peak will no longer be easily visible.

Value

A numeric vector of length max.dist+1 containing the correlation coefficients for each delay interval from 0 to max.dist.

Author(s)

Aaron Lun

References

Kharchenko PV, Tolstorukov MY and Park, PJ (2008). Design and analysis of ChIP-seq experiments for DNA-binding proteins. *Nat. Biotechnol.* 26, 1351-1359.

See Also

ccf

Examples

```
n <- 20
bamFile <- system.file("exdata", "rep1.bam", package="csaw")
par(mfrow=c(2,2))

x <- correlateReads(bamFile, max.dist=n)
plot(0:n, x, xlab="delay (bp)", ylab="ccf")

x <- correlateReads(bamFile, max.dist=n, param=readParam(dedup=TRUE))
plot(0:n, x, xlab="delay (bp)", ylab="ccf")

x <- correlateReads(bamFile, max.dist=n, cross=FALSE)
plot(0:n, x, xlab="delay (bp)", ylab="acf")
```

---

csawUsersGuide *View csaw user's guide*

Description

Finds the location of the user's guide and opens it for viewing.

Usage

```
csawUsersGuide(view=TRUE)
```

Arguments

view	logical scalar specifying whether the document should be opened

Details

Calling `vignette("csaw")` will yield a short vignette that contains little information. Instead, the user guide can be obtained with this function. The user's guide is not a true vignette as it is not generated using Sweave when the package is built. This is due to the time-consuming nature of the code when run on realistic case studies.

For non-Windows operating systems, the PDF viewer is taken from `Sys.getenv("R_PDFVIEWER")`. This can be changed to `x` by using `Sys.putenv(R_PDFVIEWER=x)`. For Windows, the default viewer will be selected to open the file. Note that for Windows, the user's guide can also be accessed from the 'Vignettes' drop-down menu in the R GUI.

Value

A character string giving the file location. If `view=TRUE`, the system's default PDF document reader is started and the user's guide is opened.

Author(s)

Aaron Lun

See Also

`system`

Examples

```
# To get the location:
csawUsersGuide(view=FALSE)
# To open in pdf viewer:
## Not run: csawUsersGuide()
```

---

detailRanges *Add annotation to ranges*

Description

Add detailed exon-based annotation to specified genomic regions.

Usage

```
detailRanges(incoming, txdb, orgdb, dist=5000,
    promoter=c(3000, 1000), max.intron=1e6)
```

Arguments

`incoming`	a `GRanges` object containing the ranges to be annotated
`txdb`	a `TranscriptDb` object for the genome of interest
`orgdb`	a genome wide annotation object for the genome of interest
`dist`	an integer scalar specifying the flanking distance to annotate
`promoter`	an integer vector of length 2, where first and second values define the promoter as some distance upstream and downstream from the TSS, respectively
`max.intron`	an integer scalar indicating the maximum distance between exons

Details

This function adds exon-based annotations to a given set of genomic regions, in the form of compact character strings specifying the features overlapping and flanking each region. The aim is to determine the genic context of empirically identified regions. This allows some basic biological interpretation of binding/marking in those regions. All neighbouring genes within a specified range are reported, rather than just the closest gene to the region.

For annotated features overlapping a region, the character string in the `overlap` output vector will be of the form `GENE|EXONS|STRAND`. The `GENE` is the gene symbol, with an alternative as `ID:XXX` for the Entrez ID if the symbol is unavailable. The `EXONS` indicate the exon or range of exons that are overlapped. The `STRAND` is, obviously, the strand on which the gene is coded. For annotated regions flanking the region within `dist`, the character string in the `left` or `right` output vectors will have an additional `DIST` value. This represents the gap between the edge of the region and the closest exon for that gene.

Exons are numbered in order of increasing start or end position for genes on the forward or reverse strands, respectively. Promoters are defined as the region of length `promoter` upstream of the gene TSS, itself defined as the start of the first exon (for genes on the forward strand) or the end of the last exon (otherwise). All promoters are marked as exon 0 for simplicity. Exon ranges in `EXON` are reported from as a comma-separated list where stretches of consecutive exons are summarized into a range. If the region overlaps an intron, it is labelled with `I` in `EXON`. No intronic overlaps are reported if there is an exonic overlap.

Note that promoter and intronic annotations are only reported for the `overlap` vector to reduce redundancy in the output. For example, it makes little sense to report that the region is both flanking and overlapping an intron. Similarly, the value of `DIST` is more relevant when it is reported to the nearest exon rather than to an intron (in which case, the distance would be zero if the intron overlaps the region). In cases where the distance is reported to the first exon, it can be used to refine the choice of `promoter`.

The `max.intron` value is necessary to deal with genes that have ambiguous locations on the genome. If a gene has exons on different chromosomes, its location is uncertain and the gene is partitioned into two sets of exons for separate processing. However, this is less obvious when the ambiguous locations belong to the same chromosome. The `max.intron` value protects against excessively large genes that may occur from considering those locations as a single transcriptional unit.

If `incoming` is missing, then the annotation will be provided directly to the user in the form of a `GRanges` object. This may be more useful when further work on the annotation is required. Exon numbers are provided in the metadata with promoters and gene bodies labelled as 0 and -1, respectively. Overlaps to introns can be identified by finding those regions that overlap with gene bodies but not with any of the corresponding exons.

Value

If `incoming` is not provided, a `GRanges` object will be returned containing ranges for the exons, promoters and gene bodies. Gene IDs, symbol and exon numbers are also stored as metadata.

If `incoming` is a `GRanges` object, a list will be returned with `overlap`, `left` and `right` elements. Each element is a character vector of length equal to the number of ranges in `incoming`. Each non-empty string records the gene symbol, the overlapped exons and the strand. For `left` and `right`, the gap between the range and the annotated feature is also included.

Author(s)

Aaron Lun

Examples

```
require(org.Mm.eg.db)
require(TxDb.Mmusculus.UCSC.mm10.knownGene)

current <- readRDS(system.file("exdata", "exrange.rds", package="csaw"))
output <- detailRanges(current, TxDb.Mmusculus.UCSC.mm10.knownGene,
    org.Mm.eg.db)
head(output$overlap)
head(output$right)
head(output$left)

output <- detailRanges(current, TxDb.Mmusculus.UCSC.mm10.knownGene,
    org.Mm.eg.db, promoter=c(2000, 1000))
head(output$overlap)
head(output$right)
head(output$left)

detailRanges(txdb=TxDb.Mmusculus.UCSC.mm10.knownGene, orgdb=org.Mm.eg.db)
```

---

extractReads *Extract reads from a BAM file*

Description

Extract reads from a BAM file with the specified parameter settings.

Usage

```
extractReads(cur.region, bam.file, param=readParam())
```

Arguments

cur.region	a GRanges object of length 1 describing the region of interest
bam.file	a character string containing the path to a sorted and indexed BAM file
param	a readParam object specifying how reads should be extracted

Details

This function extracts the reads from a BAM file overlapping a given genomic interval. The interpretation of the values in param is the same as that throughout the package. The aim is to supply the raw data for visualization, in a manner that maintains consistency with the rest of the analysis.

Note that this does not account for any read extension that might have been performed during read counting. In such cases, users are advised to expand cur.region by the extension length on each

side. Counted reads can then be extracted by identifying their extended counterparts that overlap with the original `cur.region`.

Value

A `GRanges` object is returned. If `pet="both"` in `param`, intervals are unstranded and correspond to fragments. Otherwise, strand-specific intervals that represent reads are returned.

Author(s)

Aaron Lun

See Also

`readParam`

Examples

```
bamFile <- system.file("exdata", "rep1.bam", package="csaw")
extractReads(GRanges("chrA", IRanges(100, 500)), bamFile)
extractReads(GRanges("chrA", IRanges(100, 500)), bamFile, param=readParam(dedup=TRUE))

bamFile <- system.file("exdata", "pet.bam", package="csaw")
extractReads(GRanges("chrB", IRanges(100, 500)), bamFile)
extractReads(GRanges("chrB", IRanges(100, 500)), bamFile, param=readParam(pet="both"))
extractReads(GRanges("chrB", IRanges(100, 500)), bamFile, param=readParam(pet="first"))

# Dealing with the extension length.
bamFile <- system.file("exdata", "rep1.bam", package="csaw")
ext <- 100
my.reg <- GRanges("chrA", IRanges(10, 200))
my.reg2 <- resize(my.reg, fix="center", width=width(my.reg)+2*ext)
collected <- extractReads(my.reg2, bamFile)

expanded <- resize(collected, width=ext)
strand(expanded) <- "*"
relevant <- overlapsAny(expanded, my.reg)
collected[relevant,]
```

---

getBestTest *Get the best test in a cluster*

Description

Find the test with the strongest evidence for rejection of the null in each cluster.

Usage

```
getBestTest(ids, tab, mode=c("PValue", "logCPM"),
    weight=rep(1, length(ids)))
```

Arguments

ids	an integer vector containing the cluster ID for each test
tab	a table of results with a PValue field for each test
mode	a string specifying the metric to use for selection of the best test
weight	a numeric vector of weights for each window

Details

If mode="PValue", this function identifies the test with the lowest p-value as that with the strongest evidence against the null in each cluster. The p-value of the chosen test is adjusted using the Bonferroni correction, based on the total number of tests in the parent cluster. This is necessary to obtain strong control of the family-wise error rate such that the best test can be taken from each cluster for further consideration.

The importance of each window in each cluster can be adjusted by supplying different relative weight values. Each weight is interpreted as a different threshold for each test in the cluster. Larger weights correspond to lower thresholds, i.e., less evidence is needed to reject the null for tests deemed to be more important. This may be useful for upweighting particular tests, e.g., windows containing a motif for the TF of interest.

Note the difference between this function and combineTests. The latter presents evidence for any rejections within a cluster. This function specifies the exact location of the rejection in the cluster, which may be more useful in some cases but at the cost of conservativeness. In both cases, clustering procedures such as mergeWindows can be used to identify the cluster.

If mode="logCPM", the best test is defined as that with the highest log-CPM value. This should be independent of the p-value so no adjustment is necessary. Weights are not applied here. This mode may be useful when abundance is correlated to rejection under the alternative hypothesis, e.g., picking high-abundance regions that are more likely to contain peaks.

Value

A dataframe with one row per cluster and the numeric fields best, the index for the best test in the cluster; PValue, the adjusted p-value for that test; and FDR, the q-value corresponding to the adjusted p-value.

Author(s)

Aaron Lun

References

Genovese CR, Roeder K and Wasserman L (2006). False discovery control with p-value weighting. *Biometrika* 93, 509-524.

See Also

combineTests, mergeWindows

Examples

```
ids <- round(runif(100, 1, 10))
tab <- data.frame(logFC=rnorm(100), logCPM=rnorm(100),
    PValue=rbeta(100, 1, 2))
combined <- getBestTest(ids, tab)
head(combined)

# With window weighting.
w <- round(runif(100, 1, 5))
combined <- getBestTest(ids, tab, weight=w)
head(combined)

# By logCPM.
combined <- getBestTest(ids, tab, mode="logCPM")
head(combined)
```

`getPETSizes` *Compute fragment lengths for paired-end tags*

Description

Compute the length of the sequenced fragment for each read pair in paired-end tag (PET) data.

Usage

```
getPETSizes(bam.file, param=readParam(pet="both"))
```

Arguments

`bam.file`	a character string containing the file path to a sorted and indexed BAM file
`param`	a `readParam` object containing read extraction parameters

Details

This function assembles a number of paired-end diagnostics. First, any read pairs with one or more unmapped reads are ignored. A read is only mapped if it is not removed by `dedup`, `minq`, `restrict` or `discard` in `readParam`. Otherwise, the alignment is not considered to be reliable. The total number of read pairs with one unmapped read is recorded.

Of the mapped pairs, the valid (i.e., proper) read pairs are identified. These refer to intrachromosomal read pairs where the reads with the lower and higher genomic coordinate map to the forward and reverse strand, respectively. The distance between the positions of the mapped 5' ends of the two reads must also be equal to or greater than the read lengths. Any intrachromosomal read pair that fails these criteria will be considered as improperly orientated. If the reads are on different chromosomes, the read pair will be recorded as being interchromosomal.

Each valid read pair corresponds to a DNA fragment where both ends are sequenced. The size of the fragment can be determined by calculating the distance between the 5' ends of the mapped reads. The distribution of sizes is useful for assessing the quality of the library preparation, along with all of the recorded diagnostics.

Value

A list containing:

sizes	an integer vector of fragment lengths for all valid read pairs in the library
diagnostics	an integer vector containing the total number of reads, number of singleton reads, pairs with one unmapped read, number of improperly orientated read pairs and interchromosomal pairs

Author(s)

Aaron Lun

See Also

readParam

Examples

```
bamFile <- system.file("exdata", "pet.bam", package="csaw")
out <- getPETSizes(bamFile, param=readParam(pet="both"))
out <- getPETSizes(bamFile, param=readParam(pet="both", restrict="chrA"))
out <- getPETSizes(bamFile, param=readParam(pet="both", discard=GRanges("chrA", IRanges(1, 50))))
```

---

mergeWindows *Merge windows into clusters*

Description

Uses a simple single-linkage approach to merge adjacent or overlapping windows into clusters.

Usage

```
mergeWindows(regions, tol, sign=NULL, max.width=NULL)
```

Arguments

regions	a GRanges object
tol	a numeric scalar specifying the maximum distance between adjacent windows
sign	a logical vector specifying whether each window has a positive log-FC
max.width	a numeric scalar specifying the maximum size of merged intervals

Details

Windows are merged if the gap between the end of one window and the start of the next is no greater than `tol`. Adjacent windows can then be chained together to build a cluster of windows across the linear genome. A value of zero means that the windows must be contiguous whereas negative values specify minimum overlaps.

If `sign!=NULL`, windows are only merged if they have the same sign of the log-FC and are not separated by windows with opposite log-FC values. This can be useful when summarizing adjacent regions. However, it is not recommended for routine clustering in differential analyses as the resulting clusters will not be independent of the p-value.

The `max.width` parameter can be specified to avoid the formation of excessively large clusters when many adjacent regions are present. For typical ChIP-seq datasets, suggested values range from 2000 to 10000 bp. Clusters are split if they exceed this size, albeit in a rather *ad hoc* manner. If `NULL`, no limits are placed on the maximum size.

The tolerance should reflect the minimum distance at which two regions of enrichment are considered separate. If two windows are more than `tol` apart, they *will* be placed into separate clusters. In contrast, the `max.width` value reflects the maximum distance at which two windows can be considered part of the same region.

Note that in the output, the cluster ID reported in `id` corresponds to the index of the cluster coordinates in the input `region`.

Value

A list containing `id`, an integer vector containing the cluster ID for each window; and `region`, a `GRanges` object containing the start/stop coordinates for each cluster of windows.

Author(s)

Aaron Lun

See Also

`combineTests`, `windowCounts`

Examples

```
x <- round(runif(10, 100, 1000))
gr <- GRanges(rep("chrA", 10), IRanges(x, x+40))
mergeWindows(gr, 1)
mergeWindows(gr, 10)
mergeWindows(gr, 100)
mergeWindows(gr, 100, sign=rep(c(TRUE, FALSE), 5))
```

`normalizeCounts` *Normalize counts between libraries*

Description

Calculate normalization factors or offsets using count data from multiple libraries.

Usage

```
normalizeCounts(counts, lib.sizes, type=c("scaling", "loess"),
    weighted=FALSE, dispersion=0.05, ...)
```

Arguments

`counts`	a matrix of integer counts with one column per library
`lib.sizes`	a numeric vector specifying the total number of reads per library
`type`	a character string indicating what type of normalization is to be performed
`weighted`	a logical scalar indicating whether precision weights should be used for TMM normalization
`dispersion`	a numeric scalar specifying the NB dispersion for calculation of the average count in `type="loess"`
`...`	other arguments to be passed to `calcNormFactors` for `type="scaling"`, or `loessFit` for `type="loess"`

Details

If `type="scaling"`, this function provides a convenience wrapper for the `calcNormFactors` function in the `edgeR` package. Specifically, it uses the trimmed mean of M-values (TMM) method to perform normalization. Precision weighting is turned off by default so as to avoid upweighting high-abundance regions. These are more likely to be bound and thus more likely to be differentially bound. Assigning excessive weight to such regions will defeat the purpose of trimming when normalizing the coverage of background regions.

If `type="loess"`, this function performs non-linear normalization similar to the fast loess algorithm in `normalizeCyclicLoess`. For each sample, a lowess curve is fitted to the log-counts against the log-average count. The fitted value for each bin pair is used as the generalized linear model offset for that sample. The use of the average count provides more stability than the average log-count when low counts are present for differentially bound regions.

If `lib.sizes` is not specified, the column sums of `counts` are used instead and a warning is issued. The same `lib.sizes` should be used throughout the analysis if multiple `normalizeCounts` calls are involved. This ensures that the normalization factors or offsets are comparable between calls.

Value

For `type="scaling"`, a numeric vector containing the relative normalization factors for each library.

For `type="loess"`, a numeric matrix of the same dimensions as `counts`, containing the log-based offsets for use in GLM fitting.

Author(s)

Aaron Lun

References

Robinson MD, Oshlack A (2010). A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology* 11, R25.

Ballman KV, Grill DE, Oberg AL, Therneau TM (2004). Faster cyclic loess: normalizing RNA arrays via linear models. *Bioinformatics* 20, 2778-86.

See Also

`calcNormFactors`, `loessFit`, `normalizeCyclicLoess`

Examples

```
# A trivial example
counts <- matrix(rnbinom(400, mu=10, size=20), ncol=4)
normalizeCounts(counts)
normalizeCounts(counts, lib.sizes=rep(400, 4))

# Adding undersampling
n <- 1000L
mu1 <- rep(10, n)
mu2 <- mu1
mu2[1:100] <- 100
mu2 <- mu2/sum(mu2)*sum(mu1)
counts <- cbind(rnbinom(n, mu=mu1, size=20), rnbinom(n, mu=mu2, size=20))
actual.lib.size <- rep(sum(mu1), 2)
normalizeCounts(counts, lib.sizes=actual.lib.size)
normalizeCounts(counts, logratioTrim=0.4, lib.sizes=actual.lib.size)
normalizeCounts(counts, sumTrim=0.3, lib.size=actual.lib.size)

# With and without weighting, for high-abundance spike-ins.
n <- 100000
blah <- matrix(rnbinom(2*n, mu=10, size=20), ncol=2)
tospike <- 10000
blah[1:tospike,1] <- rnbinom(tospike, mu=1000, size=20)
blah[1:tospike,2] <- rnbinom(tospike, mu=2000, size=20)
full.lib.size <- colSums(blah)

normalizeCounts(blah, weighted=TRUE, lib.sizes=full.lib.size)
normalizeCounts(blah, lib.sizes=full.lib.size)
```

```
true.value <- colSums(blah[(tospike+1):n,])/colSums(blah)
true.value <- true.value/exp(mean(log(true.value)))
true.value

# Using loess-based normalization, instead.
offsets <- normalizeCounts(counts, type="loess", lib.size=full.lib.size)
head(offsets)
offsets <- normalizeCounts(counts, type="loess", span=0.4, lib.size=full.lib.size)
offsets <- normalizeCounts(counts, type="loess", iterations=1, lib.size=full.lib.size)
```

`readParam` *readParam class and methods*

Description

Class to specify read loading parameters

Details

Each readParam object stores a number of parameters to extract reads from a BAM file. Slots are defined as:

`pet`: a character string indicating whether paired-end data is present

`max.frag`: an integer scalar, specifying the maximum fragment length corresponding to a read pair

`rescue.pairs`: a logical scalar indicating whether invalid read pairs should be rescued

`rescue.ext`: an integer scalar indicating the extension length for invalid read pairs

`dedup`: a logical scalar indicating whether marked duplicate reads should be ignored

`minq`: an integer scalar, specifying the minimum mapping quality score for an aligned read

`restrict`: a character vector containing the names of allowable chromosomes from which reads will be extracted

`discard`: a `GRanges` object containing intervals in which any alignments will be discarded

Marked duplicate reads will be removed with `dedup=TRUE`. This may be necessary when many rounds of PCR have been performed during library preparation. However, it is not recommended for routine counting as it will interfere with the downstream statistical methods. Note that the duplicate field must be set beforehand in the BAM file for this argument to have any effect.

Reads can also be filtered by their mapping quality scores if `minq` is specified at a non-`NA` value. This is generally recommended to remove low-confidence alignments. The exact threshold for `minq` will depend on the range of scores provided by the aligner. If `minq=NA`, no filtering on the score will be performed.

If `restrict` is supplied, reads will only be extracted for the specified chromosomes. This is useful to restrict the analysis to interesting chromosomes, e.g., no contigs/scaffolds or mitochondria. Conversely, if `discard` is set, a read will be removed if the corresponding alignment is wholly contained within the supplied ranges. This is useful for removing reads in repeat regions.

The `pet` parameter can take values of `"none"`, `"both"`, `"first"` or `"second"`. If `pet="none"`, reads are assumed to be single-end. Each read will then be processed on its own merits. The other settings for `pet` and the values of `max.frag` and `rescue.pairs` describe the treatment of paired-end data; see below for more information.

Additional advice for paired-end data

Reads are first extracted in a single-end manner, using the parameters described above, e.g., `discard`, `minq`, `dedup`. For `pet="both"`, the extracted reads are grouped into proper pairs. Proper pairs are defined as reads that are close together and in an inward-facing orientation. The fragment interval is defined as that bounded by the 5' ends of the two reads in a proper pair. Fragment sizes above `max.frag` are removed; use `getPETSizes` to pick an appropriate value.

If `rescue.pairs=FALSE`, only reads in proper pairs are used to construct a fragment interval. Otherwise, if `rescue.pairs=TRUE`, the function will first attempt to assign reads into proper pairs. For the remaining reads in improper pairs, the read with the higher MAPQ score will be taken and directionally extended to a length of `rescue.ext`. The extended read will then be used as the fragment interval. Similarly, any reads without a mapped mate will be extended. Note that both reads will be rescued and extended for interchromosomal read pairs.

Finally, paired-end data can also be treated as single-end data by only using one read from each pair with `pet="first"` or `"second"`. This is useful for poor-quality data where the paired-end procedure has obviously failed, e.g., with many interchromosomal read pairs or pairs with large fragment lengths. Treating the data as single-end may allow the analysis to be salvaged.

In all cases, users should ensure that each BAM file containing paired-end data is properly synchronized prior to count loading.

Constructor

`readParam(pet="none", max.frag=500, rescue.pairs=FALSE, rescue.ext=200, dedup=FALSE, minq=NA, restri`
: Creates a readParam object. Each argument is placed in the corresponding slot, with coercion into the appropriate type.

Subsetting

In the code snippets below, `x` is a readParam object.

`x$name`: Returns the value in slot `name`.

Other methods

In the code snippets below, `x` is a readParam object.

`show(x)`: Describes the parameter settings in plain English.

`reform(x, ...)`: Creates a new readParam object, based on the existing `x`. Any named arguments in `...` are used to modify the values of the slots in the new object, with type coercion as necessary.

Author(s)

Aaron Lun

See Also

`windowCounts`, `regionCounts`, `extractReads`, `getPETSizes`

Examples

```
blah <- readParam()
blah <- readParam(discard=GRanges("chrA", IRanges(1, 10)))
blah <- readParam(restrict=chr2)
blah$pet
blah$dedup

# Use reform if only some arguments need to be changed.
blah
reform(blah, dedup=TRUE)
reform(blah, pet="both", max.frag=212.0)
```

`regionCounts` *Count reads overlapping each region*

Description

Count the number of extended reads overlapping pre-specified regions

Usage

```
regionCounts(bam.files, regions, ext=100, param=readParam())
```

Arguments

`bam.files`	a character vector containing paths to sorted and indexed BAM files
`regions`	a `GRanges` object containing the regions over which reads are to be counted
`ext`	an integer scalar describing the average length of the sequenced fragment
`param`	a `readParam` object containing read extraction parameters

Details

This function simply provides a wrapper around `countOverlaps` for read counting into specified regions. It is provided so as to allow for counting with awareness of the other parameters, e.g., `ext`, `pet`. This allows users to coordinate region-based counts with those from `windowCounts`. Checking that the output `totals` are the same between the two calls is strongly recommended.

Value

A `SummarizedExperiment` object is returned containing one integer matrix. Each entry of the matrix contains the count for each library (column) at each region (row). The coordinates of each region are stored as the `rowData`. The total number of reads in each library are stored as `totals` in the `colData`.

Author(s)

Aaron Lun

See Also

countOverlaps, windowCounts, readParam, SummarizedExperiment

Examples

```
# A low filter is only used here as the examples have very few reads.
bamFiles <- system.file("exdata", c("rep1.bam", "rep2.bam"), package="csaw")
incoming <- GRanges(c(chrA, chrA, chrB, chrC),
    IRanges(c(1, 500, 100, 1000), c(200, 1000, 700, 1500)))
regionCounts(bamFiles, regions=incoming)
regionCounts(bamFiles, regions=incoming, param=readParam(restrict="chrB"))

# Loading PET data.
bamFile <- system.file("exdata", "pet.bam", package="csaw")
regionCounts(bamFile, regions=incoming, param=readParam(pet="both"))
regionCounts(bamFile, regions=incoming, param=readParam(max.frag=100,
pet="first", restrict="chrA"))
regionCounts(bamFile, regions=incoming, param=readParam(max.frag=100,
pet="both", restrict="chrA", rescue.pairs=TRUE))
```

SEmethods *Statistical wrappers for SummarizedExperiment objects*

Description

Convenience wrappers for statistical routines operating on SummarizedExperiment objects

Usage

```
normalize(object, ...)
asDGEList(object, ...)
```

Arguments

object	a SummarizedExperiment object, like that produced by windowCounts
...	other arguments to be passed to the function being wrapped

Details

Counts are extracted using the matrix corresponding to the first assay in the SummarizedExperiment object. The total library size is taken from the totals entry in the column data; warnings will be generated if this entry is not present. In the normalize method, the extracted counts and library sizes are supplied to normalizeCounts, along wth arguments in .... Similarly, the asDGEList method wraps the DGEList constructor.

Value

For normalize, either a numeric matrix or vector is returned; see normalizeCounts.

For asDGEList, a DGEList object is returned.

Author(s)

Aaron Lun

See Also

normalizeCounts, DGEList, windowCounts

Examples

```
bamFiles <- system.file("exdata", c("rep1.bam", "rep2.bam"), package="csaw")
data <- windowCounts(bamFiles, width=100, filter=1)
normalize(data)
head(normalize(data, type="loess"))

asDGEList(data)
asDGEList(data, norm.factors=c(1.11, 2.23), group=c("a", "b"))
```

---

windowCounts *Count reads overlapping each window*

Description

Count the number of extended reads overlapping a sliding window at spaced positions across the genome.

Usage

```
windowCounts(bam.files, spacing=50, width=1, ext=100, shift=0,
filter=NULL, bin=FALSE, param=readParam())
```

Arguments

bam.files	a character vector containing paths to sorted and indexed BAM files
spacing	an integer scalar specifying the distance between consecutive windows
width	an integer scalar specifying the width of the window
ext	an integer scalar describing the average length of the sequenced fragment
shift	an integer scalar specifying how much the start of each window should be shifted to the left
filter	an integer scalar for the minimum count sum across libraries for each window
bin	an integer scalar indicating whether binning should be performed
param	a readParam object containing read extraction parameters

Details

A window is defined as a genomic interval of size equal to `width`. The value of `width` can be interpreted as the width of the contact area between the DNA and protein. In practical terms, it determines the spatial resolution of the analysis. Larger windows count reads over a larger region which results in larger counts. This results in greater detection power at the cost of resolution.

By default, the first window on a chromosome starts at base position 1. This can be shifted to the left by specifying an appropriate value for `shift`. New windows are found by sliding the current window to the right by the specified `spacing`. Increasing `spacing` will reduce the frequency at which counts are extracted from the genome. This results in some loss of resolution but it may be necessary when machine memory is limited.

Each read in `bam.files` is extended by `ext` in the direction of the read to obtain a rough estimate of the fragment boundaries. The number of fragments overlapping the window for each library is then counted for each window position. For single-end data, the value of `ext` can be estimated using `correlateReads` or from fragment length diagnostics during library preparation. For paired-end data, the fragment boundaries can be computed exactly for proper pairs, by specifying `pet="both"` in `readParam`. If `rescue.pairs=TRUE`, improperly paired reads are salvaged by directional extension to `ext`.

Windows will be removed if the count sum across all libraries is below `filter`. This reduces the memory footprint of the output by not returning empty or near-empty windows, which are usually uninteresting anyway. If `filter=NULL`, the count sum filter threshold is automatically defined as the number of libraries multiplied by 5.

If `bin` is set, settings are internally adjusted so that all reads are counted into non-overlapping adjacent bins of size `width`. Specifically, `spacing` is set to `bin` and `filter` is set to 1. Only the 5' end of each read or left-most position of each fragment (for paired-end data) is used in counting.

Value

A `SummarizedExperiment` object is returned containing one integer matrix. Each entry of the matrix contains the count for each library (column) at each window (row). The coordinates of each window are stored as the `rowData`. The total number of reads in each library are stored as `totals` in the `colData`.

Author(s)

Aaron Lun

See Also

`correlateReads`, `readParam`, `SummarizedExperiment`

Examples

```
# A low filter is only used here as the examples have very few reads.
bamFiles <- system.file("exdata", c("rep1.bam", "rep2.bam"), package="csaw")
windowCounts(bamFiles, filter=1)
windowCounts(bamFiles, width=100, filter=1)
windowCounts(bamFiles, spacing=100, filter=1)
```

```
# Loading PET data.
bamFile <- system.file("exdata", "pet.bam", package="csaw")
windowCounts(bamFile, param=readParam(pet="both"), filter=1)
windowCounts(bamFile, param=readParam(pet="first"), filter=1)
windowCounts(bamFile, param=readParam(max.frag=100, pet="both"), filter=1)
windowCounts(bamFile, param=readParam(max.frag=100, pet="both", restrict="chrA"), filter=1)

# Running rescues of PET data (use -1 to coerce single-endedness ).
windowCounts(bamFile, param=readParam(max.frag=50, pet="both", rescue.pairs=TRUE), filter=1)
```

Index