Package 'derfinder'

April 9, 2015

Type Package

Title Annotation-agnostic differential expression analysis of RNA-seq data at base-pair resolution

Version 1.0.10

Date 2014-11-22

Depends R(>= 3.1.1)

Imports AnnotationDbi (>= 1.27.9), BiocParallel, bumphunter (>= 1.3.3), derfinderHelper (>= 1.0.0), GenomeInfoDb (>= 1.2.2), GenomicAlignments, GenomicFeatures, GenomicFiles, GenomicRanges (>= 1.17.40), Hmisc, IRanges (>= 1.99.28), qvalue, Rsamtools, rtracklayer, S4Vectors (>= 0.2.3)

Suggests biovizBase, devtools (>= 1.6), derfinderData (>= 0.99.0), ggplot2, knitcitations (>= 1.0.1), knitr (>= 1.6), knitrBootstrap (>= 0.9.0), rmarkdown (>= 0.3.3), testthat, TxDb.Hsapiens.UCSC.hg19.knownGene

VignetteBuilder knitr

Description Annotation-agnostic differential expression analysis of RNA-seq data by calculating F-statistics at base-pair resolution

License Artistic-2.0

LazyData true

URL https://github.com/lcolladotor/derfinder

BugReports https://github.com/lcolladotor/derfinder/issues

biocViews DifferentialExpression, Sequencing, RNASeq, Software

Author Leonardo Collado-Torres [aut, cre],
Alyssa C. Frazee [aut],
Andrew E. Jaffe [aut],
Jeffrey T. Leek [aut, ths]

Maintainer Leonardo Collado-Torres <lcollado@jhu.edu>

R topics documented:

`derfinder-package` *Fast differential expression analysis of RNA-seq data at base-pair resolution*

Description

Fast differential expression analysis of RNA-seq data at base-pair resolution from multiple samples. The analysis pipeline involves loading the sample BAM files using rawFiles and loadCoverage, preprocessing the data by using preprocessCoverage, calculating the F-statistics (while adjusting for some confounders) using makeModels and calculateStats, calculating the p-values and finding the regions of interest using calculatePvalues, and finally annotating them using annotateNearest from the bumphunter package.

Author(s)

Leonardo Collado-Torres <lcollado@jhu.edu>

References

Frazee, A. C., Sabunciyan, S., Hansen, K. D., Irizarry, R. A. & Leek, J. T. Differential expression analysis of RNA-seq data at single-base resolution. Biostatistics (2014). doi:10.1093/biostatistics/kxt053

Rafael A. Irizarry, Martin Aryee, Hector Corrada Bravo, Kasper D. Hansen and Harris A. Jaffee. bumphunter: Bump Hunter. R package version 1.1.10.

`advancedArg` *List advanced arguments*

Description

Find in GitHub the documentation for the advanced arguments of a given function.

Usage

```
advancedArg(fun, package = "derfinder", browse = interactive())
```

Arguments

`fun`	The name of a function(s) that has advanced arguments in `package`.
`package`	The name of the package where the function is stored. Only 'derfinder', 'derfinderPlot', and 'regionReport' are accepted.
`browse`	Whether to open the URLs in a browser.

Details

If you are interested on the default options used for functions that run on multiple cores, check https://github.com/lcolladotor/derfinder/blob/master/R/utils.R Note that in general, SnowParam is more memory efficient than link[BiocParallel]MulticoreParam. If you so desire, use your favorite cluster type by specifying `BPPARAM`.

Value

A vector of URLs with the GitHub search queries.

Author(s)

Leonardo Collado-Torres

Examples

```
## Open the advanced argument docs for loadCoverage()
if(interactive()) {
    advancedArg(loadCoverage)
} else {
    (advancedArg(loadCoverage, browse = FALSE))
}
```

---

analyzeChr *Run the derfinder analysis on a chromosome*

Description

This is a major wrapper for running several key functions from this package. It is meant to be used after loadCoverage has been used for a specific chromosome. The steps run include makeModels, preprocessCoverage, calculateStats, calculatePvalues and annotateNearest.

Usage

```
analyzeChr(chr, coverageInfo, models, cutoffPre = 5, cutoffFstat = 1e-08,
  cutoffType = "theoretical", nPermute = 1, seeds = as.integer(gsub("-",
  "", Sys.Date())) + seq_len(nPermute), groupInfo, subject = "hg19",
  writeOutput = TRUE, runAnnotation = TRUE, lowMemDir = file.path(chr,
  "chunksDir"), ...)
```

Arguments

chr	Used for naming the output files when writeOutput=TRUE and for annotateNearest.
coverageInfo	A list containing a DataFrame –$coverage– with the coverage data and a logical Rle –$position– with the positions that passed the cutoff. This object is generated using loadCoverage.
models	The output from makeModels.
cutoffPre	This argument is passed to preprocessCoverage (cutoff).
cutoffFstat	This is used to determine the cutoff argument of calculatePvalues and it's behaviour is determined by cutoffType.
cutoffType	If set to empirical, the cutoffFstat (example: 0.99) quantile is used via quantile. If set to theoretical, the theoretical cutoffFstats (example: 1e-08) is calculated via qf. If set to manual, cutoffFstats is passed to calculatePvalues without any other calculation.
nPermute	The number of permutations. Note that for a full chromosome, a small amount (10) of permutations is sufficient. If set to 0, no permutations are performed and thus no null regions are used, however, the $regions component is created.
seeds	An integer vector of length nPermute specifying the seeds to be used for each permutation. If NULL no seeds are used.

groupInfo	A factor specifying the group membership of each sample that can later be used with the plotting functions in the derfinderPlot package.
subject	This argument is passed to annotateNearest. Note that only hg19 works right now.
writeOutput	If TRUE, output Rdata files are created at each step inside a directory with the chromosome name (example: 'chr21' if chrnum=21). One Rdata file is created for each component described in the return section.
runAnnotation	If TRUE annotateNearest is run. Otherwise this step is skipped.
lowMemDir	If specified, each chunk is saved into a separate Rdata file under lowMemDir and later loaded in fstats.apply when running calculateStats and calculatePvalues. Using this option helps reduce the memory load as each fork in bplapply loads only the data needed for the chunk processing. The downside is a bit longer computation time due to input/output.
...	Arguments passed to other methods and/or advanced arguments.

Details

If you are working with data from an organism different from 'Homo sapiens' specify so by setting the global 'species' and 'chrsStyle' options. For example: `options(species = arabidopsis_thaliana)` `options(chrsStyle = NCBI)`

Value

If `returnOutput=TRUE`, a list with six components:

timeinfo The wallclock timing information for each step.

optionsStats The main options used when running this function.

coveragePrep The output from preprocessCoverage.

fstats The output from calculateStats.

regions The output from calculatePvalues.

annotation The output from annotateNearest.

These are the same components that are written to Rdata files if `writeOutput=TRUE`.

Author(s)

Leonardo Collado-Torres

See Also

makeModels, preprocessCoverage, calculateStats, calculatePvalues, annotateNearest

Examples

```
## Collapse the coverage information
collapsedFull <- collapseFullCoverage(list(genomeData$coverage),
    verbose = TRUE)

## Calculate library size adjustments
sampleDepths <- sampleDepth(collapsedFull, probs = c(0.5), nonzero=TRUE,
    verbose=TRUE)

## Build the models
groupInfo <- genomeInfo$pop
adjustvars <- data.frame(genomeInfo$gender)
models <- makeModels(sampleDepths, testvars=groupInfo, adjustvars=adjustvars)

## Analyze the chromosome
results <- analyzeChr(chr=21, coverageInfo=genomeData, models=models,
    cutoffFstat=1, cutoffType=manual, groupInfo=groupInfo, mc.cores=1,
    writeOutput=FALSE, returnOutput=TRUE, method=regular)
names(results)
```

---

annotateRegions *Assign genomic states to regions*

Description

This function takes the regions found in calculatePvalues and assigns them genomic states contructed with makeGenomicState. The main workhorse functions are countOverlaps and findOverlaps.

Usage

```
annotateRegions(regions, genomicState, annotate = TRUE, ...)
```

Arguments

regions	The $regions output from calculatePvalues.
genomicState	A GRanges object created with makeGenomicState. It can be either the genomicState$fullGenome or genomicState$codingGenome component.
annotate	If TRUE then the regions are annotated by the genomic state. Otherwise, only the overlaps between the regions and the genomic states are computed.
...	Arguments passed to other methods and/or advanced arguments.

Details

You might want to specify arguments such as minoverlap to control how the overlaps are determined. See findOverlaps for further details.

Value

A list with elements `countTable` and `annotationList` (only if `annotate=TRUE`).

countTable This is a data.frame with the number of overlaps from the regions vs the genomic states with one type per column. For example, if `fullOrCoding=full` then the columns are `exon`, `intragenic` and `intron`.

annotationList This is a `GRangesList` with the genomic states that overlapped with the regions. The names of this `GRangesList` correspond to the region index in `regions`.

Author(s)

Andrew Jaffe, Leonardo Collado-Torres

See Also

makeGenomicState, calculatePvalues

Examples

```
## Annotate regions, first two only
annotatedRegions <- annotateRegions(regions=genomeRegions$regions[1:2],
    genomicState=genomicState$fullGenome, minoverlap=1)
annotatedRegions
```

---

calculatePvalues *Calculate p-values and identify regions*

Description

First, this function finds the regions of interest according to specified cutoffs. Then it permutes the samples and re-calculates the F-statistics. The area of the statistics from these segments are then used to calculate p-values for the original regions.

Usage

```
calculatePvalues(coveragePrep, models, fstats, nPermute = 1L,
  seeds = as.integer(gsub("-", "", Sys.Date())) + seq_len(nPermute), chr,
  cutoff = quantile(fstats, 0.99), significantCut = c(0.05, 0.1),
  lowMemDir = NULL, ...)
```

Arguments

`coveragePrep`	A list with `$coverageProcessed`, `$mclapplyIndex`, and `$position` normally generated using preprocessCoverage.
`models`	A list with `$mod` and `$mod0` normally generated using makeModels.
`fstats`	A numerical Rle with the F-statistics normally generated using calculateStats.

nPermute	The number of permutations. Note that for a full chromosome, a small amount (10) of permutations is sufficient. If set to 0, no permutations are performed and thus no null regions are used, however, the $regions component is created.
seeds	An integer vector of length nPermute specifying the seeds to be used for each permutation. If NULL no seeds are used.
chr	A single element character vector specifying the chromosome name. This argument is passed to findRegions.
cutoff	F-statistic cutoff to use to determine segments.
significantCut	A vector of length two specifiying the cutoffs used to determine significance. The first element is used to determine significance for the p-values and the second element is used for the q-values.
lowMemDir	The directory where the processed chunks are saved when using preprocessCoverage with a specified lowMemDir.
...	Arguments passed to other methods and/or advanced arguments.

Value

A list with four components:

regions is a GRanges with metadata columns given by findRegions with the additional metadata column pvalues: p-value of the region calculated via permutations of the samples; qvalues: the qvalues calculated using qvalue; significant: whether the p-value is less than 0.05 (by default); significantQval: whether the q-value is less than 0.10 (by default). It also includes the mean coverage of the region (mean from the mean coverage at each base calculated in preprocessCoverage). Furthermore, if groupInfo was not NULL in preprocessCoverage, then the group mean coverage is calculated as well as the log 2 fold change (using group 1 as the reference).

nullStats is a numeric Rle with the mean of the null statistics by segment.

nullWidths is a numeric Rle with the length of each of the segments in the null distribution. The area can be obtained by multiplying the absolute nullstats by the corresponding lengths.

nullPermutation is a Rle with the permutation number from which the null region originated from.

Author(s)

Leonardo Collado-Torres

See Also

findRegions, fstats.apply, qvalue

Examples

```
## Collapse the coverage information
collapsedFull <- collapseFullCoverage(list(genomeData$coverage),
    verbose = TRUE)

## Calculate library size adjustments
```

```
sampleDepths <- sampleDepth(collapsedFull, probs=c(0.5), verbose = TRUE)

## Build the models
group <- genomeInfo$pop
adjustvars <- data.frame(genomeInfo$gender)
models <- makeModels(sampleDepths, testvars = group, adjustvars = adjustvars)

## Preprocess the data
## Automatic chunksize used to then compare 1 vs 4 cores in the do not run
## section
prep <- preprocessCoverage(genomeData, groupInfo = group, cutoff = 0,
    scalefac = 32, chunksize = NULL, colsubset = NULL, mc.cores = 4)

## Get the F statistics
fstats <- genomeFstats

## We recommend determining the cutoff to use based on the F-distribution
## although you could also based it on the observed F-statistics.

## In this example we use a low cutoff used for illustrative purposes
cutoff <- 1

## Calculate the p-values and define the regions of interest.
regsWithP <- calculatePvalues(prep, models, fstats, nPermute=1, seeds=1,
    chr = chr21, cutoff = cutoff, mc.cores = 1, method = regular)
regsWithP

## Not run:
## Calculate again, but with 10 permutations instead of just 1
regsWithP <- calculatePvalues(prep, models, fstats, nPermute=10, seeds=1:10,
    chr=chr21, cutoff=cutoff, mc.cores=2, method=regular)

## Check that they are the same as the previously calculated regions
library(testthat)
expect_that(regsWithP, equals(genomeRegions))

## Histogram of the theoretical p-values by region
hist(pf(regsWithP$regions$value, df1-df0, n-df1), main=Distribution
    original p-values by region, freq=FALSE)

## Histogram of the permutted p-values by region
hist(regsWithP$regions$pvalues, main=Distribution permutted p-values by
    region, freq=FALSE)

## MA style plot
library(ggplot2)
ma <- data.frame(mean=regsWithP$regions$meanCoverage,
    log2FoldChange=regsWithP$regions$log2FoldChangeYRIvsCEU)
ggplot(ma, aes(x=log2(mean), y=log2FoldChange)) + geom_point() +
    ylab(Fold Change (log2)) + xlab(Mean coverage (log2)) +
    labs(title=MA style plot)

## Annotate the results
```

```
library(bumphunter)
annotation <- annotateNearest(regsWithP$regions, hg19)
head(annotation)


## End(Not run)
```

`calculateStats` *Calculate F-statistics at base pair resolution from a loaded BAM files*

Description

After defining the models of interest (see makeModels) and pre-processing the data (see preprocessCoverage), use calculateStats to calculate the F-statistics at base-pair resolution.

Usage

```
calculateStats(coveragePrep, models, lowMemDir = NULL, ...)
```

Arguments

`coveragePrep`	A list with `$coverageProcessed`, `$mclapplyIndex`, and `$position` normally generated using preprocessCoverage.
`models`	A list with `$mod` and `$mod0` normally generated using makeModels.
`lowMemDir`	The directory where the processed chunks are saved when using preprocessCoverage with a specified `lowMemDir`.
`...`	Arguments passed to other methods and/or advanced arguments.

Value

A numeric Rle with the F-statistics per base pair that passed the cutoff.

Author(s)

Leonardo Collado-Torres

See Also

makeModels, preprocessCoverage

Examples

```
## Collapse the coverage information
collapsedFull <- collapseFullCoverage(list(genomeData$coverage),
    verbose = TRUE)

## Calculate library size adjustments
sampleDepths <- sampleDepth(collapsedFull, probs=c(0.5), verbose = TRUE)

## Build the models
group <- genomeInfo$pop
adjustvars <- data.frame(genomeInfo$gender)
models <- makeModels(sampleDepths, testvars = group, adjustvars = adjustvars)

## Preprocess the data
prep <- preprocessCoverage(genomeData, cutoff = 0, scalefac = 32,
    chunksize=1e3, colsubset=NULL)

## Run the function
fstats <- calculateStats(prep, models, verbose=TRUE, method=regular)
fstats

## Not run:
## Compare vs pre-packaged F-statistics
library(testthat)
expect_that(fstats, is_equivalent_to(genomeFstats))

## End(Not run)
```

`coerceGR` *Coerce the coverage to a GRanges object for a given sample*

Description

Given the output of fullCoverage, coerce the coverage to a GRanges object.

Usage

```
coerceGR(sample, fullCov, ...)
```

Arguments

`sample`	The name or integer index of the sample of interest to coerce to a `GRanges` object.
`fullCov`	A list where each element is the result from loadCoverage used with `returnCoverage = TRUE`. Can be generated using fullCoverage.
`...`	Arguments passed to other methods and/or advanced arguments.

Value

A GRanges object with `score` metadata vector containing the coverage information for the specified sample. The ranges reported are only those for regions of the genome with coverage greater than zero.

Author(s)

Leonardo Collado-Torres

See Also

GRanges

Examples

```
## Create a small fullCov object with data only for chr21
fullCov <- list(chr21 = genomeDataRaw)

## Coerce to a GRanges the first sample
gr <- createBwSample(ERR009101, fullCov = fullCov,
    seqlengths = c(chr21 = 48129895))

## Explore the output
gr
```

`collapseFullCoverage` *Collapse full coverage information for efficient quantile computations*

Description

For a given data set this function collapses the full coverage information for each sample from all the chromosomes. The resulting information per sample is the number of bases with coverage 0, 1, etc. It is similar to using table() on a regular vector. This information is then used by sampleDepth for calculating the sample depth adjustments. The data set can loaded to R using (see fullCoverage) and optionally filtered using filterData.

Usage

```
collapseFullCoverage(fullCov, colsubset = NULL, save = FALSE, ...)
```

Arguments

`fullCov`	A list where each element is the result from loadCoverage used with `cutoff=NULL`. Can be generated using fullCoverage.
`colsubset`	Which columns of `coverageInfo$coverage` to use.
`save`	If `TRUE`, the result is saved as 'collapsedFull.Rdata'.
`...`	Arguments passed to other methods and/or advanced arguments.

Value

A list with one element per sample. Then per sample, a list with two vector elements: `values` and `weights`. The first one is the coverage value and the second one is the number of bases with that value.

Author(s)

Leonardo Collado-Torres

See Also

fullCoverage, sampleDepth

Examples

```
## Collapse the coverage information for the filtered data
collapsedFull <- collapseFullCoverage(list(genomeData),
    verbose=TRUE)
collapsedFull

## Not run:
## You can also collapsed the raw data
collapsedFullRaw <- collapseFullCoverage(list(genomeDataRaw), verbose=TRUE)

## End(Not run)
```

---

coverageToExon *Extract coverage information for exons*

Description

This function extracts the coverage information calculated by fullCoverage for a set of exons determined by makeGenomicState. The underlying code is similar to getRegionCoverage with additional tweaks for calculating RPKM values.

Usage

```
coverageToExon(fullCov = NULL, genomicState, L = NULL, returnType = "raw",
  files = NULL, ...)
```

Arguments

`fullCov`	A list where each element is the result from loadCoverage used with `returnCoverage = TRUE`. Can be generated using fullCoverage. Alternatively, specify `files` to extract the coverage information from the regions of interest. This can be helpful if you do not wish to store `fullCov` for memory reasons.
`genomicState`	A GRanges object created with makeGenomicState. It can be either the `genomicState$fullGenome` or `genomicState$codingGenome` component.

L	The width of the reads used.
returnType	If raw, then the raw coverage information per exon is returned. If rpkm, RPKM values are calculated for each exon.
files	A character vector with the full path to the sample BAM files (or BigWig files). The names are used for the column names of the DataFrame. Check rawFiles for constructing files. files can also be a BamFileList object created with BamFileList or a BigWigFileList object created with BigWigFileList.
...	Arguments passed to other methods and/or advanced arguments.

Details

Parallelization is used twice. First, it is used by strand. Second, for processing the exons by chromosome. So there is no gain in using mc.cores greater than the maximum of the number of strands and number of chromosomes.

If fullCov is NULL and files is specified, this function will attempt to read the coverage from the files. Note that if you used 'totalMapped' and 'targetSize' before, you will have to specify them again to get the same results.

See also advancedArg with fun=loadCoverage for other details.

Value

A matrix (nrow = number of exons in genomicState corresponding to the chromosomes in fullCov, ncol = number of samples) with the number of reads (or RPKM) per exon. The row names correspond to the row indexes of genomicState$fullGenome (if fullOrCoding=full) or genomicState$codingGenome (if fullOrCoding=coding).

Author(s)

Andrew Jaffe, Leonardo Collado-Torres

See Also

fullCoverage, getRegionCoverage

Examples

```
## Obtain fullCov object
fullCov <- list(21=genomeDataRaw$coverage)

## Use only the first two exons
smallGenomicState <- genomicState
smallGenomicState$fullGenome <- smallGenomicState$fullGenome[
    which(smallGenomicState$fullGenome$theRegion == exon)[1:2] ]

## Finally, get the coverage information for each exon
exonCov <- coverageToExon(fullCov=fullCov,
    genomicState=smallGenomicState$fullGenome, L=36)
```

createBw *Export coverage to BigWig files*

Description

Using output from fullCoverage, export the coverage from all the samples to BigWig files using createBwSample.

Usage

```
createBw(fullCov, path = ".", keepGR = TRUE, ...)
```

Arguments

fullCov	A list where each element is the result from loadCoverage used with returnCoverage = TRUE. Can be generated using fullCoverage.
path	The path where the BigWig files will be created.
keepGR	If TRUE, the GRanges objects created by coerceGR grouped into a GRangesList are returned. Otherwise they are discarded.
...	Arguments passed to other methods and/or advanced arguments.

Details

Use at most one core per chromosome.

Value

If keepGR = TRUE, then a GRangesList with the output for coerceGR for each of the samples.

Author(s)

Leonardo Collado-Torres

See Also

GRangesList, export, createBwSample, coerceGR

Examples

```
## Create a small fullCov object with data only for chr21
fullCov <- list(chr21 = genomeDataRaw)

## Keep only 2 samples
fullCov$chr21$coverage <- fullCov$chr21$coverage[c(1, 31)]

## Create the BigWig files for all samples in a test dir
dir.create(createBw-example)
bws <- createBw(fullCov, createBw-example)
```

```
## Explore the output
bws

## First sample
bws[[1]]

## Note that if a sample has no bases with coverage > 0, the GRanges object
## is empty and no BigWig file is created for that sample.
bws[[2]]
```

---

createBwSample *Create a BigWig file with the coverage information for a given sample*

Description

Given the output of fullCoverage, this function coerces the coverage to a GRanges object using coerceGR and then exports the coverage to a BigWig file using export.

Usage

```
createBwSample(sample, path = ".", fullCov, keepGR = TRUE, ...)
```

Arguments

sample	The name or integer index of the sample of interest to coerce to a GRanges object.
path	The path where the BigWig file will be created.
fullCov	A list where each element is the result from loadCoverage used with returnCoverage = TRUE. Can be generated using fullCoverage.
keepGR	If TRUE, the GRanges object created by coerceGR is returned. Otherwise it is discarded.
...	Arguments passed to other methods and/or advanced arguments.

Value

Creates a BigWig file with the coverage information (regions with coverage greater than zero) for a given sample. If keepGR it returns the output from coerceGR.

Author(s)

Leonardo Collado-Torres

See Also

GRanges, export, linkcoerceGR

Examples

```
## Create a small fullCov object with data only for chr21
fullCov <- list(chr21 = genomeDataRaw)

## Create a BigWig for the first sample in a test directory
dir.create(createBwSample-example)
bw <- createBwSample(ERR009101, createBwSample-example,
    fullCov = fullCov, seqlengths = c(chr21 = 48129895))

## Explore the output
bw
```

---

extendedMapSeqlevels *Change naming style for a set of sequence names*

Description

If available, use the information from GenomeInfoDb for your species of interest to map the sequence names from the style currently used to another valid style. For example, for Homo sapiens map '2' (NCBI style) to 'chr2' (UCSC style). If the information from GenomeInfoDb is not available, the original sequence names will be returned.

Usage

```
extendedMapSeqlevels(seqnames, style = getOption("chrsStyle", "UCSC"),
  species = getOption("species", "homo_sapiens"), currentStyle = NULL, ...)
```

Arguments

seqnames	A character vector with the sequence names.
style	A single character vector specifying the naming style to use for renaming the sequence names.
species	A single character vector with the species of interest: it has to match the valid species names supported in GenomeInfoDb. See `names(GenomeInfoDb::genomeStyles())`. If `species = NULL`, a guess will be made using the available information in GenomeInfoDb.
currentStyle	A single character vector with the currently used naming style. If `NULL`, a guess will be made from the naming styles supported by `species`.
...	Arguments passed to other methods and/or advanced arguments.

Details

This function is inspired from mapSeqlevels with the difference that it will return the original sequence names if the species, current naming style or target naming style are not supported in GenomeInfoDb.

Value

A vector of sequence names using the specified naming `style`.

Author(s)

L. Collado-Torres

Examples

```
## Without guessing any information
extendedMapSeqlevels(2, UCSC, Homo sapiens, NCBI)

## Guessing the current naming style
extendedMapSeqlevels(2, UCSC, Homo sapiens)

## Guess species and current style
extendedMapSeqlevels(2, NCBI)

## Guess species while supplying the current style.
## Probably an uncommon use-case
extendedMapSeqlevels(2, NCBI, currentStyle = TAIR10)

## Sequence names are unchanged when using an unsupported species
extendedMapSeqlevels(seq2, NCBI, toyOrganism)

## Not run:
## Set global species and style option
options(chrsStyle = UCSC)
options(species = homo_sapiens)

## Run using global options
extendedMapSeqlevels(2)

## End(Not run)
```

---

filterData *Filter the positions of interest*

Description

For a group of samples this function reads the coverage information for a specific chromosome directly from the BAM files. It then merges them into a DataFrame and removes the bases that do not pass the cutoff. This is a helper function for loadCoverage and preprocessCoverage.

Usage

```
filterData(data, cutoff = NULL, index = NULL, filter = "one",
  totalMapped = NULL, targetSize = 8e+07, ...)
```

Arguments

data	Either a list of Rle objects or a DataFrame with the coverage information.
cutoff	The base-pair level cutoff to use. It's behavior is controlled by filter.
index	A logical Rle with the positions of the chromosome that passed the cutoff. If NULL it is assumed that this is the first time using filterData and thus no previous index exists.
filter	Has to be either one (default) or mean. In the first case, at least one sample has to have coverage above cutoff. In the second case, the mean coverage has to be greater than cutoff.
totalMapped	The total number of reads mapped for each sample. Providing this data adjusts the coverage to reads in targetSize library prior to filtering. By default, to reads per 80 million reads.
targetSize	The target library size to adjust the coverage to. Used only when totalMapped is specified.
...	Arguments passed to other methods and/or advanced arguments.

Details

If cutoff is NULL then the data is grouped into DataFrame without applying any cutoffs. This can be useful if you want to use loadCoverage to build the coverage DataFrame without applying any cutoffs for other downstream purposes like plotting the coverage values of a given region. You can always specify the colsubset argument in preprocessCoverage to filter the data before calculating the F statistics.

Value

A list with up to three components.

coverage is a DataFrame object where each column represents a sample. The number of rows depends on the number of base pairs that passed the cutoff and the information stored is the coverage at that given base. Included only when returnCoverage = TRUE.

position is a logical Rle with the positions of the chromosome that passed the cutoff.

meanCoverage is a numeric Rle with the mean coverage at each base. Included only when returnMean = TRUE.

Author(s)

Leonardo Collado-Torres

See Also

loadCoverage, preprocessCoverage

Examples

```
## Construct some toy data
library(IRanges)
x <- Rle(round(runif(1e4, max=10)))
y <- Rle(round(runif(1e4, max=10)))
z <- Rle(round(runif(1e4, max=10)))
DF <- DataFrame(x, y, z)

## Filter the data
filt1 <- filterData(DF, 5)
filt1

## Filter again but only using the first two samples
filt2 <- filterData(filt1$coverage[, 1:2], 5, index=filt1$position)
filt2
```

`findRegions` *Find non-zero regions in a Rle*

Description

Find genomic regions for which a numeric vector is above (or below) predefined thresholds. In other words, this function finds the candidate Differentially Expressed Regions (candidate DERs). This is similar to regionFinder and is a helper function for calculatePvalues.

Usage

```
findRegions(position = NULL, fstats, chr, oneTable = TRUE,
  maxClusterGap = 300L, cutoff = quantile(fstats, 0.99), segmentIR = NULL,
  ...)
```

Arguments

`position`	A logical Rle of genomic positions. This is generated in loadCoverage. Note that it gets updated in preprocessCoverage if `colsubset` is not `NULL`.
`fstats`	A numeric Rle with the F-statistics. Usually obtained using calculateStats.
`chr`	A single element character vector specifying the chromosome name.
`oneTable`	If `TRUE` only one GRanges is returned. Otherwise, a GRangesList with two components is returned: one for the regions with positive values and one for the negative values.
`maxClusterGap`	This determines the maximum gap between candidate DERs. It should be greater than `maxRegionGap` (0 by default).
`cutoff`	Threshold applied to the `fstats` used to determine the #' regions.
`segmentIR`	An IRanges object with the genomic positions that are potentials DERs. This is used in calculatePvalues to speed up permutation calculations.
`...`	Arguments passed to other methods and/or advanced arguments.

Details

regionFinder adapted to Rle world.

Value

Either a GRanges or a GRangesList as determined by `oneTable`. Each of them has the following metadata variables.

value The mean of the values of `y` for the given region.

area The absolute value of the sum of the values of `y` for the given region.

indexStart The start position of the region in terms of the index for `y`.

indexEnd The end position of the region in terms of the index for `y`.

cluster The cluser ID.

clusterL The total length of the cluster.

Author(s)

Leonardo Collado-Torres

References

Rafael A. Irizarry, Martin Aryee, Hector Corrada Bravo, Kasper D. Hansen and Harris A. Jaffee. bumphunter: Bump Hunter. R package version 1.1.10.

See Also

calculatePvalues

Examples

```
## Preprocess the data
prep <- preprocessCoverage(genomeData, cutoff=0, scalefac=32, chunksize=1e3,
    colsubset=NULL)

## Get the F statistics
fstats <- genomeFstats

## Find the regions
regs <- findRegions(prep$position, fstats, chr21, verbose=TRUE)
regs

## Not run:
## Once you have the regions you can proceed to annotate them
annotation <- bumphunter::annotateNearest(regs, hg19)
annotation

## End(Not run)
```

`fullCoverage` *Load the unfiltered coverage information from a group of BAM files and a list of chromosomes*

Description

For a group of samples this function reads the coverage information for several chromosomes directly from the BAM files. Per chromosome, it merges the unfiltered coverage by sample into a DataFrame. The end result is a list with one such DataFrame objects per chromosome.

Usage

```
fullCoverage(files, chrs, bai = NULL, chrlens = NULL, outputs = NULL,
  cutoff = NULL, ...)
```

Arguments

`files`	A character vector with the full path to the sample BAM files (or BigWig files). The names are used for the column names of the DataFrame. Check rawFiles for constructing `files`. `files` can also be a `BamFileList` object created with BamFileList or a `BigWigFileList` object created with BigWigFileList.
`chrs`	The chromosome of the files to read. The format has to match the one used in the input files.
`bai`	The full path to the BAM index files. If `NULL` it is assumed that the BAM index files are in the same location as the BAM files and that they have the .bai extension. Ignored if `files` is a `BamFileList` object.
`chrlens`	The chromosome lengths in base pairs. If it's `NULL`, the chromosome length is extracted from the BAM files. Otherwise, it should have the same length as `chrs`.
`outputs`	This argument is passed to the `output` argument of loadCoverage. If `NULL` or `auto` it is then recycled.
`cutoff`	This argument is passed to filterData.
`...`	Arguments passed to other methods and/or advanced arguments.

Value

A list with one element per chromosome.

Each element is a DataFrame with the coverage information produced by loadCoverage.

Author(s)

Leonardo Collado-Torres

See Also

loadCoverage, filterData

Examples

```
datadir <- system.file(extdata, genomeData, package=derfinder)
files <- rawFiles(datadir=datadir, samplepatt=*accepted_hits.bam$,
    fileterm=NULL)
## Shorten the column names
names(files) <- gsub(_accepted_hits.bam, , names(files))

## Read and filter the data, only for 1 file
fullCov <- fullCoverage(files=files[1], chrs=c(21, 22))
fullCov

## Not run:
## You can then use filterData() to filter the data if you want to.
## Use bplapply() if you want to do so with multiple cores as shown below.
library(BiocParallel)
p <- SnowParam(2L, outfile = Sys.getenv(SGE_STDERR_PATH))
bplapply(fullCov, function(x) {
    library(derfinder); filterData(x, cutoff=0) }, BPPARAM = p)

## End(Not run)
```

---

genomeData *Genome samples processed data*

Description

10kb region from chr21 processed for 31 RNA-seq samples described in genomeInfo. The TopHat BAM files are included in the package and this is the output of loadCoverage applied to it. For more information check the example of loadCoverage.

Format

A list with two components.

coverage is a DataFrame object where each column represents a sample.

position is a logical Rle with the positions of the chromosome that passed a cutoff of 0.

References

1. Pickrell JK, Marioni JC, Pai AA, Degner JF, Engelhardt BE, Nkadori E, Veyrieras J-B, Stephens M, Gilad Y, Pritchard JK. Understanding mechanisms underlying human gene expression variation with RNA sequencing. Nature 2010 Apr.

2. Montgomery SB, Sammeth M, Gutierrez-Arcelus M, Lach RP, Ingle C, Nisbett J, Guigo R, Dermitzakis ET. Transcriptome genetics using second generation sequencing in a Caucasian population. Nature 2010 Mar.

See Also

loadCoverage, genomeInfo

genomeDataRaw *Genome samples processed data*

Description

10kb region from chr21 processed for 31 RNA-seq samples described in genomeInfo. The TopHat BAM files are included in the package and this is the output of loadCoverage applied to it with `cutoff=NULL`. For more information check the example of loadCoverage.

Format

A list with two components.

coverage is a DataFrame object where each column represents a sample.

position is `NULL` because no bases were filtered.

References

1. Pickrell JK, Marioni JC, Pai AA, Degner JF, Engelhardt BE, Nkadori E, Veyrieras J-B, Stephens M, Gilad Y, Pritchard JK. Understanding mechanisms underlying human gene expression variation with RNA sequencing. Nature 2010 Apr.

2. Montgomery SB, Sammeth M, Gutierrez-Arcelus M, Lach RP, Ingle C, Nisbett J, Guigo R, Dermitzakis ET. Transcriptome genetics using second generation sequencing in a Caucasian population. Nature 2010 Mar.

See Also

loadCoverage, genomeInfo

genomeFstats *F-statistics for the example data*

Description

Calculated F-statistics for a 10kb region from chr21 processed for 31 RNA-seq samples described in genomeInfo. For more information check the example of calculateStats.

Format

A numeric Rle of length 1434 with the calculated F-statistics as exemplified in calculateStats.

See Also

calculateStats

genomeInfo *Genome samples information*

Description

Information for the 31 samples downloaded from the Short Read Archive from studies comparing CEU and YRI populations. This data is used to specify the adjustment variables in calculateStats. The data is sorted according to the BAM files identifiers. Reads were 36bp long.

Format

A data.frame with 5 columns:

run The short name used to identify the sample BAM file.

library.layout Whether it was a single-end library or a paired-end library.

hapmap.id The HapMap identifier of the person sequenced. Note that some were sequenced more than once.

gender Whether the person sequence is a female or a male.

pop The population the person belongs to.

Details

The samples are from:

10 unrelated females from the YRI population.

5 unrelated females from the CEU population.

5 unrelated males (unrelated to the females too) from the CEU population.

References

1. Pickrell JK, Marioni JC, Pai AA, Degner JF, Engelhardt BE, Nkadori E, Veyrieras J-B, Stephens M, Gilad Y, Pritchard JK. Understanding mechanisms underlying human gene expression variation with RNA sequencing. Nature 2010 Apr.

2. Montgomery SB, Sammeth M, Gutierrez-Arcelus M, Lach RP, Ingle C, Nisbett J, Guigo R, Dermitzakis ET. Transcriptome genetics using second generation sequencing in a Caucasian population. Nature 2010 Mar.

See Also

genomeData, calculateStats

genomeRegions *Candidate DERs for example data*

Description

Candidate Differentially Expressed Regions (DERs) for the example data. For more information check calculatePvalues.

Format

A list with four components.

regions a GRanges object with the candidate DERs.

nullStats a numeric Rle with the mean F-statistics for the null DERs found from the permutations.

nullWidths an integer Rle with the width of each null candidate DER.

nullPermutation an integer Rle with the permutation number for each candidate DER. It identifies which permutation cycle created the null candidate DER.

See Also

calculatePvalues

genomicState *Genomic State for Hsapiens.UCSC.hg19.knownGene*

Description

Pre-computed genomic state for Hsapiens UCSC hg19 knownGene annotation built using makeGenomicState for TxDb.Hsapiens.UCSC.hg19.knownGene version 2.14.0. The object has been subset for chr21 only.

Format

A GRangesList with two components.

fullGenome classifies each region as either being exon, intron or intragenic.

codingGenome classfies the regions as being promoter, exon, intro, 5UTR, 3UTR or intragenic.

See Also

makeGenomicState

`getRegionCoverage` *Extract coverage information for a set of regions*

Description

This function extracts the raw coverage information calculated by fullCoverage at each base for a set of regions found with calculatePvalues. It can further calculate the mean coverage per sample for each region.

Usage

```
getRegionCoverage(fullCov = NULL, regions, totalMapped = NULL,
  targetSize = 8e+07, files = NULL, ...)
```

Arguments

`fullCov`	A list where each element is the result from loadCoverage used with `returnCoverage = TRUE`. Can be generated using fullCoverage. Alternatively, specify `files` to extract the coverage information from the regions of interest. This can be helpful if you do not wish to store `fullCov` for memory reasons.
`regions`	The `$regions` output from calculatePvalues. It is important that the seqlengths information is provided.
`totalMapped`	The total number of reads mapped for each sample. Providing this data adjusts the coverage to reads in `targetSize` library. By default, to reads per 80 million reads.
`targetSize`	The target library size to adjust the coverage to. Used only when `totalMapped` is specified.
`files`	A character vector with the full path to the sample BAM files (or BigWig files). The names are used for the column names of the DataFrame. Check rawFiles for constructing `files`. `files` can also be a `BamFileList` object created with BamFileList or a `BigWigFileList` object created with BigWigFileList.
`...`	Arguments passed to other methods and/or advanced arguments.

Details

When `fullCov` is the output of loadCoverage with `cutoff` non-NULL, getRegionCoverage assumes that the regions come from the same data. Meaning that filterData was not used again. This ensures that the regions are a subset of the data available in `fullCov`.

If `fullCov` is `NULL` and `files` is specified, this function will attempt to read the coverage from the files. Note that if you used 'totalMapped' and 'targetSize' before, you will have to specify them again to get the same results.

See also advancedArg with `fun=loadCoverage` for other details.

You should use at most one core per chromosome.

Value

a list of data.frame where each data.frame has the coverage information (nrow = width of region, ncol = number of samples) for a given region. The names of the list correspond to the region indexes in `regions`

Author(s)

Andrew Jaffe, Leonardo Collado-Torres

See Also

fullCoverage, calculatePvalues

Examples

```
## Obtain fullCov object
fullCov <- list(21=genomeDataRaw$coverage)

## Assign chr lengths using hg19 information, use only first two regions
library(GenomicRanges)
data(hg19Ideogram, package = biovizBase, envir = environment())
regions <- genomeRegions$regions[1:2]
seqlengths(regions) <- seqlengths(hg19Ideogram)[names(seqlengths(regions))]

## Finally, get the region coverage
regionCov <- getRegionCoverage(fullCov=fullCov, regions=regions)
```

---

loadCoverage *Load the coverage information from a group of BAM files*

Description

For a group of samples this function reads the coverage information for a specific chromosome directly from the BAM files. It then merges them into a DataFrame and removes the bases that do not pass the cutoff.

Usage

```
loadCoverage(files, chr, cutoff = NULL, filter = "one", chrlen = NULL,
  output = NULL, bai = NULL, ...)
```

Arguments

`files`	A character vector with the full path to the sample BAM files (or BigWig files). The names are used for the column names of the DataFrame. Check rawFiles for constructing `files`. `files` can also be a BamFileList, BamFile, BigWigFileList, or BigWigFile object.

chr	Chromosome to read. Should be in the format matching the one used in the raw data.
cutoff	This argument is passed to filterData.
filter	Has to be either one (default) or mean. In the first case, at least one sample has to have coverage above cutoff. In the second case, the mean coverage has to be greater than cutoff.
chrlen	The chromosome length in base pairs. If it's NULL, the chromosome length is extracted from the BAM files.
output	If NULL then no output is saved in disk. If auto then an automatic name is constructed using UCSC names (chrXCovInfo.Rdata for example). If another character is specified, then that name is used for #' the output file.
bai	The full path to the BAM index files. If NULL it is assumed that the BAM index files are in the same location as the BAM files and that they have the .bai extension. Ignored if files is a BamFileList object or if inputType==BigWig.
...	Arguments passed to other methods and/or advanced arguments.

Details

The ... argument can be used to control which alignments to consider when reading from BAM files. See scanBamFlag.

Parallelization for loading the data in chunks is used only used when tilewidth is specified. You may use up to one core per tile.

If you set the advanced argument drop.D = TRUE, bases with CIGAR string "D" (deletion from reference) will be excluded from the base-level coverage calculation.

If you are working with data from an organism different from 'Homo sapiens' specify so by setting the global 'species' and 'chrsStyle' options. For example: options(species = arabidopsis_thaliana) options(chrsStyle = NCBI)

Value

A list with two components.

coverage is a DataFrame object where each column represents a sample. The number of rows depends on the number of base pairs that passed the cutoff and the information stored is the coverage at that given base.

position is a logical Rle with the positions of the chromosome that passed the cutoff.

Author(s)

Leonardo Collado-Torres, Andrew Jaffe

Examples

```
datadir <- system.file(extdata, genomeData, package=derfinder)
files <- rawFiles(datadir = datadir, samplepatt = *accepted_hits.bam$,
    fileterm = NULL)
## Shorten the column names
```

```
names(files) <- gsub(_accepted_hits.bam, , names(files))

## Read and filter the data, only for 2 files
dataSmall <- loadCoverage(files = files[1:2], chr = 21, cutoff = 0)

## Not run:
## Export to BigWig files
createBw(list(chr21 = dataSmall))

## Load data from BigWig files
dataSmall.bw <- loadCoverage(c(ERR009101 = ERR009101.bw, ERR009102 =
    ERR009102.bw), chr = chr21)

## Compare them
mapply(function(x, y) { x - y }, dataSmall$coverage, dataSmall.bw$coverage)

## Note that the only difference is the type of Rle (integer vs numeric) used
## to store the data.


## End(Not run)
```

`makeGenomicState` *Obtain the genomic state per region from annotation*

Description

This function summarizes the annotation contained in a TxDb at each given base of the genome based on annotated transcripts. It groups contiguous base pairs classified as the same type into regions.

Usage

```
makeGenomicState(txdb, chrs = c(1:22, "X", "Y"), ...)
```

Arguments

`txdb`	A TxDb object.
`chrs`	The names of the chromosomes to use as denoted in the `txdb` object. Check isActiveSeq.
`...`	Arguments passed to other methods and/or advanced arguments.

Value

A `GRangesList` object with two elements: `fullGenome` and `codingGenome`. Both have metadata information for the type of region (theRegion), transcript IDs (tx_id), transcript name (tx_name), and gene ID (gene_id). `fullGenome` classifies each region as either being exon, intron or intragenic. `codingGenome` classfies the regions as being promoter, exon, intro, 5UTR, 3UTR or intragenic.

Author(s)

Andrew Jaffe, Leonardo Collado-Torres

See Also

TxDb

Examples

```
## Load the example data base from the GenomicFeatures vignette
library(GenomicFeatures)
samplefile <- system.file(extdata, hg19_knownGene_sample.sqlite,
    package=GenomicFeatures)
txdb <- loadDb(samplefile)

## Generate genomic state object, only for chr6
sampleGenomicState <- makeGenomicState(txdb, chrs=chr6)
## Not run:
## Create the GenomicState object for Hsapiens.UCSC.hg19.knownGene
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## Creating this GenomicState object takes around 8 min for all chrs and
## around 30 secs for chr21
GenomicState.Hsapiens.UCSC.hg19.knownGene.chr21 <-
    makeGenomicState(txdb=txdb, chrs=chr21)

## For convinience, this object is already included in derfinder
library(testthat)
expect_that(GenomicState.Hsapiens.UCSC.hg19.knownGene.chr21,
   is_equivalent_to(genomicState))

## Hsapiens ENSEMBL GRCh37
library(GenomicFeatures)
## Can take several minutes and speed will depend on your internet speed
xx <- makeTxDbPackageFromBiomart(version = 0.99, maintainer = Your Name,
    author=Your Name)
txdb <- loadDb(file.path(TxDb.Hsapiens.BioMart.ensembl.GRCh37.p11, inst,
    extdata, TxDb.Hsapiens.BioMart.ensembl.GRCh37.p11.sqlite))

## Creating this GenomicState object takes around 13 min
GenomicState.Hsapiens.ensembl.GRCh37.p11 <- makeGenomicState(txdb=txdb,
    chrs=c(1:22, X, Y))

## Save for later use
save(GenomicState.Hsapiens.ensembl.GRCh37.p11,
    file=GenomicState.Hsapiens.ensembl.GRCh37.p11.Rdata)

## End(Not run)
```

`makeModels` *Build model matrices for differential expression*

Description

Builds the model matrices for testing for differential expression by comparing a model with a grouping factor versus one without it. It adjusts for the confounders specified and the median coverage of each sample. The resulting models can be used in calculateStats.

Usage

```
makeModels(sampleDepths, testvars, adjustvars = NULL, testIntercept = FALSE)
```

Arguments

`sampleDepths`	Per sample library size adjustments calculated with sampleDepth.
`testvars`	A vector or matrix specifying the variables to test. For example, a factor with the group memberships when testing for differences across groups. It's length should match the number of columns used from `coverageInfo$coverage`.
`adjustvars`	Optional matrix of adjustment variables (e.g. measured confounders, output from SVA, etc.) to use in fitting linear models to each nucleotide. These variables have to be specified by sample and the number of rows must match the number of columns used. It will also work if it is a vector of the correct length.
`testIntercept`	If `TRUE` then `testvars` is ignored and mod0 will contain the column medians and any adjusting variables specified, but no intercept.

Value

A list with two components.

mod The alternative model matrix.

mod0 The null model matrix.

Author(s)

Leonardo Collado-Torres

See Also

sampleDepth, calculateStats

Examples

```
## Collapse the coverage information
collapsedFull <- collapseFullCoverage(list(genomeData$coverage),
    verbose=TRUE)

## Calculate library size adjustments
sampleDepths <- sampleDepth(collapsedFull, probs=c(0.5), nonzero=TRUE,
    verbose=TRUE)

## Build the models
group <- genomeInfo$pop
adjustvars <- data.frame(genomeInfo$gender)
models <- makeModels(sampleDepths, testvars=group, adjustvars=adjustvars)
names(models)
models
```

`mergeResults` *Merge results from different chromosomes*

Description

This function merges the results from running analyzeChr on several chromosomes and assigns genomic states using annotateRegions. It re-calculates the p-values and q-values using the pooled areas from the null regions from all chromosomes. Once the results have been merged, `derfinderReport::generateReport` can be used to generate an HTML report of the results. The `derfinderReport` package is available at https://github.com/lcolladotor/derfinderReport.

Usage

```
mergeResults(chrs = c(1:22, "X", "Y"), prefix = ".",
  significantCut = c(0.05, 0.1), genomicState, minoverlap = 20,
  mergePrep = FALSE, ...)
```

Arguments

`chrs`	The chromosomes of the files to be merged.
`prefix`	The main data directory path, which can be useful if analyzeChr is used for several parameters and the results are saved in different directories.
`significantCut`	A vector of length two specifiying the cutoffs used to determine significance. The first element is used to determine significance for the p-values and the second element is used for the q-values just like in calculatePvalues.
`genomicState`	A GRanges object created with makeGenomicState. It can be either the `genomicState$fullGenome` or `genomicState$codingGenome` component.
`minoverlap`	Determines the mininum overlap needed when annotating regions with annotateRegions.
`mergePrep`	If `TRUE` the output from preprocessCoverage is merged.
`...`	Arguments passed to other methods and/or advanced arguments.

Details

If you want to calculate the FWER, supply `optionsStats` which #' is produced by analyzeChr.

Value

Seven Rdata files.

fullFstats.Rdata Full F-statistics from all chromosomes in a list of Rle objects.

fullTime.Rdata Timing information from all chromosomes.

fullNullSummary.Rdata A DataFrame with the null region information: statistic, width, chromosome and permutation identifier. It's ordered by the statistics

fullRegions.Rdata GRanges object with regions found and with full annotation from annotateNearest. Note that the column `strand` from annotateNearest is renamed to `annoStrand` to comply with GRanges specifications.

fullCoveragePrep.Rdata A list with the pre-processed coverage data from all chromosomes.

fullAnnotatedRegions.Rdata A list as constructed in annotateRegions with the assigned genomic states.

optionsMerge.Rdata A list with the options used when merging the results. Used in `derfinderReport::generateReport`.

Author(s)

Leonardo Collado-Torres

See Also

analyzeChr, calculatePvalues, annotateRegions

Examples

```
## The output will be saved in the generateReport-example directory
dir.create(generateReport-example, showWarnings = FALSE, recursive = TRUE)

## For convenience, the derfinder output has been pre-computed
file.copy(system.file(file.path(extdata, chr21), package=derfinder,
mustWork=TRUE), generateReport-example, recursive=TRUE)

## Merge the results from the different chromosomes. In this case, theres
## only one: chr21
mergeResults(chrs=21, prefix=generateReport-example,
genomicState=genomicState$fullGenome)

## Not run:
## You can then explore the wallclock time spent on each step
load(file.path(generateReport-example, fullRegions.Rdata))

## Process the time info
time <- lapply(fullTime, function(x) data.frame(diff(x)))
time <- do.call(rbind, time)
colnames(time) <- sec
```

```
time$sec <- as.integer(round(time$sec))
time$min <- time$sec / 60
time$chr <- paste0(chr, gsub(\\..*, , rownames(time)))
time$step <- gsub(.*\\., , rownames(time))
rownames(time) <- seq_len(nrow(time))

## Make plot
library(ggplot2)
ggplot(time, aes(x=step, y=min, colour=chr)) + geom_point() +
    labs(title=Wallclock time by step) +
    scale_colour_discrete(limits=chrs) +
    scale_x_discrete(limits=names(fullTime[[1]])[-1]) + ylab(Time (min)) +
    xlab(Step)

## End(Not run)
```

`preprocessCoverage` *Transform and split the data*

Description

This function takes the coverage data from loadCoverage, scales the data, does the log2 transformation, and splits it into appropriate chunks for using calculateStats.

Usage

```
preprocessCoverage(coverageInfo, groupInfo = NULL, cutoff = 5,
  colsubset = NULL, lowMemDir = NULL, ...)
```

Arguments

`coverageInfo`	A list containing a DataFrame –`$coverage`– with the coverage data and a logical Rle –`$position`– with the positions that passed the cutoff. This object is generated using loadCoverage.
`groupInfo`	A factor specifying the group membership of each sample. If `NULL` no group mean coverages are calculated. If the factor has more than one level, the first one will be used to calculate the log2 fold change in calculatePvalues.
`cutoff`	The base-pair level cutoff to use. It's behavior is controlled by `filter`.
`colsubset`	Optional vector of column indices of `coverageInfo$coverage` that denote samples you wish to include in analysis.
`lowMemDir`	If specified, each chunk is saved into a separate Rdata file under `lowMemDir` and later loaded in fstats.apply when running calculateStats and calculatePvalues. Using this option helps reduce the memory load as each fork in bplapply loads only the data needed for the chunk processing. The downside is a bit longer computation time due to input/output.
`...`	Arguments passed to other methods and/or advanced arguments.

Details

If chunksize is NULL, then mc.cores is used to determine the chunksize. This is useful if you want to split the data so each core gets the same amount of data (up to rounding).

Computing the indexes and using those for mclapply reduces memory copying as described by Ryan Thompson and illustrated in approach #4 at http://lcolladotor.github.io/2013/11/14/Reducing-memory-overhead-when-using-mclapply

If lowMemDir is specified then $coverageProcessed is NULL and $mclapplyIndex is a vector with the chunk identifiers.

Value

A list with five components.

coverageProcessed contains the processed coverage information in a DataFrame object. Each column represents a sample and the coverage information is scaled and log2 transformed. Note that if colsubset is not NULL the number of columns will be less than those in coverageInfo$coverage. The total number of rows depends on the number of base pairs that passed the cutoff and the information stored is the coverage at that given base. Further note that filterData is re-applied if colsubset is not NULL and could thus lead to fewer rows compared to coverageInfo$coverage.

mclapplyIndex is a list of logical Rle objects. They contain the partioning information according to chunksize.

position is a logical Rle with the positions of the chromosome that passed the cutoff.

meanCoverage is a numeric Rle with the mean coverage at each filtered base.

groupMeans is a list of Rle objects containing the mean coverage at each filtered base calculated by group. This list has length 0 if groupInfo=NULL.

Author(s)

Leonardo Collado-Torres

See Also

filterData, loadCoverage, calculateStats

Examples

```
## Split the data and transform appropriately before using calculateStats()
dataReady <- preprocessCoverage(genomeData, cutoff = 0, scalefac = 32,
    chunksize = 1e3, colsubset = NULL, verbose = TRUE)
names(dataReady)
dataReady
```

rawFiles *Construct full paths to a group of raw input files*

Description

For a group of samples this function creates the list of paths to the raw input files which can then be used in loadCoverage. The raw input files are either BAM files or BigWig files.

Usage

```
rawFiles(datadir = NULL, sampledirs = NULL, samplepatt = NULL,
  fileterm = "accepted_hits.bam")
```

Arguments

datadir	The main directory where each of the sampledirs is a sub-directory of datadir.
sampledirs	A character vector with the names of the sample directories. If datadir is NULL it is then assumed that sampledirs specifies the full path to each sample.
samplepatt	If specified and sampledirs is set to NULL, then the directories matching this pattern in datadir (set to . if it's set to NULL) are used as the sample directories.
fileterm	Name of the BAM or BigWig file used in each sample. By default it is set to accepted_hits.bam since that is the automatic name generated when aligning with TopHat. If NULL it is then ignored when reading the rawfiles. This can be useful if all the raw files are stored in a single directory.

Details

This function can also be used to identify a set of BigWig files.

Value

A vector with the full paths to the raw files and sample names stored as the vector names.

Author(s)

Leonardo Collado-Torres

See Also

loadCoverage

Examples

```
## Get list of BAM files included in derfinder
datadir <- system.file(extdata, genomeData, package=derfinder)
files <- rawFiles(datadir=datadir, samplepatt=*accepted_hits.bam$,
    fileterm=NULL)
files
```

`regionMatrix` *Identify regions data by a coverage filter and get a count matrix*

Description

Given a set of un-filtered coverage data (see fullCoverage), create candidate regions by applying a cutoff on the coverage values, and obtain a count matrix where the number of rows corresponds to the number of candidate regions and the number of columns corresponds to the number of samples. The values are the mean coverage for a given sample for a given region.

Usage

```
regionMatrix(fullCov, cutoff = 5, filter = "mean", L, runFilter = TRUE,
  returnBP = TRUE, ...)
```

Arguments

`fullCov`	A list where each element is the result from loadCoverage used with `returnCoverage = TRUE`. Can be generated using fullCoverage. If `runFilter = FALSE`, then `returnMean = TRUE` must have been used.
`cutoff`	The base-pair level cutoff to use. It's behavior is controlled by `filter`.
`filter`	Has to be either `one` (default) or `mean`. In the first case, at least one sample has to have coverage above `cutoff`. In the second case, the mean coverage has to be greater than `cutoff`.
`L`	The width of the reads used.
`runFilter`	This controls whether to run filterData or not. If set to `FALSE` then `returnMean = TRUE` must have been used to create each element of `fullCov`.
`returnBP`	If `TRUE`, returns `$bpCoverage` explained below.
`...`	Arguments passed to other methods and/or advanced arguments.

Details

This function uses several other derfinder-package functions. Inspect the code if interested.

You should use at most one core per chromosome.

Value

A list with one entry per chromosome. Then per chromosome, a list with three components.

regions A set of regions based on the coverage filter cutoff as returned by findRegions.

bpCoverage A list with one element per region. Each element is a matrix with numbers of rows equal to the number of base pairs in the region and number of columns equal to the number of samples. It contains the base-level coverage information for the regions. Only returned when `returnBP = TRUE`.

coverageMatrix A matrix with the mean coverage by sample for each candidate region.

Author(s)

Leonardo Collado-Torres

Examples

```
## Create some toy data
library(IRanges)
x <- Rle(round(runif(1e4, max=10)))
y <- Rle(round(runif(1e4, max=10)))
z <- Rle(round(runif(1e4, max=10)))
fullCov <- list(chr21 = DataFrame(x, y, z))

## Calculate a proxy of library size
libSize <- sapply(fullCov$chr21, sum)

## Run region matrix normalizing the coverage
regionMat <- regionMatrix(fullCov = fullCov, maxRegionGap = 10L,
    maxClusterGap = 300L, L = 36, totalMapped = libSize, targetSize = 4e4)

## Not run:
## You can alternatively use filterData() on fullCov to reduce the required
## memory before using regionMatrix(). This can be useful when mc.cores > 1
filteredCov <- lapply(fullCov, filterData, returnMean=TRUE, filter=mean,
    cutoff=5, totalMapped = libSize, targetSize = 4e4)
regionMat2 <- regionMatrix(filteredCov, maxRegionGap = 10L,
    maxClusterGap = 300L, L = 36, runFilter=FALSE)
identical(regionMat2, regionMat)

## End(Not run)
```

---

sampleDepth *Calculate adjustments for library size*

Description

For a given data set calculate the per-sample coverage adjustments. Hector Corrada's group proposed calculating the sum of the coverage for genes below a given sample quantile. In this function, we calculate the sample quantiles of interest by sample, and then the sum of the coverage for bases below or equal to quantiles of interest. The resulting values are transformed log2(x + scalefac) to avoid very large numbers that could potentially affect the stability of the F-statistics calculation. The sample coverage adjustments are then used in makeModels for constrcuing the null and alternative models.

Usage

```
sampleDepth(collapsedFull, probs = c(0.5, 1), scalefac = 32, ...)
```

Arguments

collapsedFull	The full coverage data collapsed by sample as produced by collapseFullCoverage.
probs	Number(s) between 0 and 1 representing the quantile(s) of interest. For example, 0.5 is the median.
scalefac	Number added to the sample coverage adjustments before the log2 transformation.
...	Arguments passed to other methods and/or advanced arguments.

Value

A matrix (vector of `length(probs) == 1`) with the library size depth adjustments per sample to be used in makeModels. The number of rows corresponds to the number of quantiles used for the sample adjustments.

Author(s)

Leonardo Collado-Torres

References

Paulson, J. N., Stine, O. C., Bravo, H. C. & Pop, M. Differential abundance analysis for microbial marker-gene surveys. Nat. Methods (2013). doi:10.1038/nmeth.2658

See Also

collapseFullCoverage, makeModels

Examples

```
## Collapse the coverage information
collapsedFull <- collapseFullCoverage(list(genomeData$coverage),
    verbose=TRUE)

## Calculate library size adjustments
sampleDepths <- sampleDepth(collapsedFull, probs=c(0.5, 1), verbose=TRUE)
sampleDepths
```

Index