Package 'Genominator'

April 16, 2019

Version 1.36.0

Title Analyze, manage and store genomic data

Description Tools for storing, accessing, analyzing and visualizing genomic data.

Author James Bullard, Kasper Daniel Hansen

Maintainer James Bullard <bullard@stat.berkeley.edu>

Depends R (>= 2.10), methods, RSQLite, DBI (>= 0.2-5), BiocGenerics (>= 0.1.0), IRanges (>= 2.5.27), GenomeGraphs

Imports graphics, stats, utils

Suggests biomaRt, ShortRead, yeastRNASeq

Collate Genominator.R importAndManage.R annotation.R goodnessOfFit.R plotRegion.R coverage.R diffExp.R emZero.R primingWeights.R

License Artistic-2.0

LazyLoad yes

biocViews Infrastructure

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

git_branch RELEASE_3_8

git_last_commit d9658ef

git_last_commit_date 2018-10-30

Date/Publication 2019-04-15

R topics documented:

`Genominator-package` *Data backend for Genomic data*

Description

This package implements a data backend for genomic data, ie. data mapped to a genome with chromosome, location and possibly strand information. The data is stored in an SQLite database.

We are primarily using the package for analyzing mRNA-Seq data generated from a Solexa machine, but have also used it in part of a larger project incorporating Solexa data, tiling array data from various experiments and cDNA sequencing data.

It interfaces well with the GenomeGraphs package.

Read the package vignettes for extensive use cases.

To cite this package, please see the output of `citation("Genominator")`.

Author(s)

James Bullard <bullard@stat.berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

`addPrimingWeights` *Adding priming weights to an AlignedRead object.*

Description

This function adds priming weights to an AlignedRead object.

Usage

```
addPrimingWeights(aln, weights = NULL, overwrite = FALSE, ...)
```

Arguments

`aln`	An object of class `AlignedRead`.
`weights`	A vector of weights as produced by `computePrimingWeights`.
`overwrite`	A logical, will a `weights` entry in the `alignData` of the `aln` argument be overwritten?
`...`	These arguments are passed to `computePrimingWeights` and are only used if `weights` are `NULL`.

Details

If the weights are not supplied, the weights are calculated using the `aln` object itself.

Value

An object of class `AlignedRead` with a `weights` component in its `alignData` slot.

Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>.

References

Hansen, K. D., Brenner, S. E. and Dudoit, S (2010) Biases in Illumina transcriptome sequencing caused by random hexamer priming. Nucleic Acids Res, doi:10.1093/nar/gkq224

See Also

`computePrimingWeights` and the extended example in the 'Working with ShortRead' vignette.

Examples

```
if(require(ShortRead)) {
  bwt.file <- system.file("extdata", "bowtie", "s_1_aligned_bowtie.txt",
                          package="ShortRead")
  aln <- readAligned(bwt.file, type = "Bowtie")
  weights <- computePrimingWeights(aln, weightsLength = 2L)
  aln <- addPrimingWeights(aln, weights = weights)
  head(alignData(aln)$weights)
}
```

---

aggregateExpData *Collapse data into unique entries*

Description

Collapses data based on unique combinations of values in a set of columns, by default adding a column giving counts of data entries with a particular combination.

Usage

```
aggregateExpData(expData, by = getIndexColumns(expData),
  tablename = NULL, deleteOriginal = FALSE, overwrite = FALSE,
  verbose = getOption("verbose"), colname = "counts",
  aggregator = paste("count(", by[1], ")", sep = ""))
```

Arguments

expData	An object of class ExpData.
by	Vector containing column names used to define unique entries.
tablename	Name of database table to write output data to.
deleteOriginal	Logical indicating whether original database table in ExpData object should be deleted.
overwrite	Logical indicating whether database table referred to in tablename argument should be overwritten.
verbose	Logical indicating whether details should be printed.
colname	Name of column for recording aggregation output (by default, counts).
aggregator	SQLite code used for aggregating. See Details for more information.

Details

By default this function counts instances of data entries with a particular combination of the values in the set of columns indicated in the by argument. Other SQLite commands can be indicated using the aggregator argument.

Value

Returns an ExpData object.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information.

Examples

```
N  <- 10000 # the number of observations.
df <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                 location = sample(1:1000, size = N, replace = TRUE),
                 strand = sample(c(1L,-1L), size = N, replace = TRUE))
eDataRaw <- aggregateExpData(importToExpData(df, dbFilename = tempfile(),
                             tablename = "ex_tbl", overwrite = TRUE))
```

---

applyMapped *Apply a function over mapped data.*

Description

Apply a function over each element of a list containing data subsets, organized by annotation, with an additional argument for the annotation element associated with the list item.

Usage

```
applyMapped(mapped, annoData, FUN, bindAnno = FALSE)
```

Arguments

mapped	A list of data subsets, typically the return value of a call to splitByAnnotation. Names should correspond to names of annoData object.
annoData	A data frame which must contain the columns chr, start, end and strand which specifies annotation regions of interest.
FUN	A function of two arguments, the first being an element of mapped, the second being the corresponding element of annoData.
bindAnno	Logical indicating whether annotation information should be included in the output. If TRUE it assumes the output of FUN is conformable into a data.frame.

Value

If bindAnno is FALSE, returns a list containing the output of FUN for each element of the original mapped argument. If bindAnno is TRUE, returns a data frame, containing annotation information and output of FUN.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
s <- splitByAnnotation(ed, yeastAnno[1:100,],
                       what = getColnames(ed, all = FALSE),
                       ignoreStrand = TRUE, addOverStrand = TRUE)

## compute the per-base rate for this dataset.
applyMapped(s, yeastAnno, function(dta, anno) {
   colSums(dta, na.rm = TRUE)/(anno$end - anno$start + 1)
}, bindAnno = TRUE)[1:4,]
```

---

collapseExpData *Combine multiple data sets*

Description

This function takes a dataset with data from multiple experiments, and combines the data across multiple experiments according to a user-specified function.

Usage

```
collapseExpData(expData, tablename = NULL,
  what = getColnames(expData, all = FALSE), groups = "COL",
  collapse = c("sum", "avg", "weighted.avg"), overwrite = FALSE,
  deleteOriginal = FALSE, verbose = getOption("verbose"))
```

Arguments

expData	An object of class ExpData.
tablename	Name of database table to write output data to.
what	Data columns to apply collapse function to.
groups	Vector of length what indicating how columns should be grouped when applying collapse function.
collapse	Function to apply to grouped columns.
overwrite	Logical indicating whether database referred to in tablename argument should be overwritten.
deleteOriginal	Logical indicating whether original database in ExpData object should be deleted.
verbose	Logical indicating whether details should be printed.

Details

This function can be thought of as similar to tapply, operating over the entries in the data set, applying the function specified in the collapse argument, grouping the data as indicated in the groups argument.

Value

Returns an object of class ExpData.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
nd <- importToExpData(head(ed, -1), dbFilename = tempfile(),
                      tablename = "collapsed")
head(nd)
cd <- collapseExpData(nd, tablename = "bio", overwrite = TRUE,
                      groups = c("mut", "mut", "wt", "wt"))

head(cd)
```

computeCoverage *Compute effort-coverage values*

Description

Compute fraction coverage obtained for a certain degree of sequencing effort.

Usage

```
computeCoverage(expData, annoData,
  cutoff = function(x, anno, group) { x > 10 },
  effort = seq(1e+05, 5e+07, length = 20),
  smooth = function(probs) { probs },
  groups = rep("ALL", length(what)),
  what = getColnames(expData, all = FALSE),
  totals = summarizeExpData(expData, what = what, verbose = verbose),
  ignoreStrand = FALSE, verbose = getOption("verbose"), ...)
```

Arguments

expData	An ExpData object.
annoData	A data frame which must contain the columns chr, start, end and strand which specifies annotation regions of interest.
cutoff	A predicate which determines when a region of annotation has been "sequenced". This function takes three arguments x = number of reads in region, anno = the annotation description of the region, group = the group it is in.
effort	Effort is a vector of how much sequencing has been done.
smooth	A function which takes as input the vector of probabilities and must return the probabilities.
groups	The different groups for which to calculate coverage.
what	The different columns, must be the same length as the groups.
totals	The lane totals, or some other totals. This allows us to estimate the sampling probability vector.
ignoreStrand	Whether or not to add over strands.
verbose	Do you want to see output.
...	Extra argument passed to cutoff.

Details

This argument is pretty general as different ways of specifying the arguments allows one to compute "coverage" under a lot of different definitions.

Value

Returns an object of class genominator.coverage. Pretty much you'll want to call plot on this object.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See the plot.genominator.coverage for the plotting method and the Genominator vignette for details.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
a <- computeCoverage(ed, yeastAnno, effort = 2^(5:18),
                     cutoff = function(x, ...) x > 1, smooth = FALSE)
names(a)
```

---

computePrimingWeights *Compute weights to correct for random hexamer priming.*

Description

This function computes weights used to correct for random hexamer priming, as per the reference.

Usage

```
computePrimingWeights(aln, biasedIndex = 1:2, unbiasedIndex = 24:29,
  weightsLength = 7L, returnSep = FALSE)
```

Arguments

aln	An object of class AlignedRead.
biasedIndex	A vector of start positions for the biased k-mers.
unbiasedIndex	A vector of start positions for the unbiased k-mers.
weightsLength	The length of the k-mers.
returnSep	A logical indicating whether the numerator and denominator of the weights should be return or the weights themselves.

Value

If returnSep = FALSE a named vector of weights. Otherwise a list with two elements giving the numerator (p_unbiased) and the denominator (p_biased) of the weights.

Author(s)

Kasper Daniel Hansen <khansen@jhsph.edu>.

References

Hansen, K. D., Brenner, S. E. and Dudoit, S (2010) Biases in Illumina transcriptome sequencing caused by random hexamer priming. Nucleic Acids Res, doi:10.1093/nar/gkq224

See Also

addPrimingWeights and the extended example in the 'Working with ShortRead' vignette.

Examples

```
if(require(ShortRead)) {
  bwt.file <- system.file("extdata", "bowtie", "s_1_aligned_bowtie.txt",
                          package="ShortRead")
  aln <- readAligned(bwt.file, type = "Bowtie")
  weights <- computePrimingWeights(aln, weightsLength = 2L)
  aln <- addPrimingWeights(aln, weights = weights)
  head(alignData(aln)$weights)
}
```

ExpData-class *Class "ExpData"*

Description

A class for representing experimental data organized along a genome.

Objects from the Class

The preferred way to construct objects of class ExpData is to use the constructor function ExpData(dbFilename = "file
= "tablename")

Slots

dbFilename: A "character" containing the filename of the SQLite database.

tablename: A "character" containing the tablename of the relevant SQLite table.

indexColumns: A "character", listing which columns (and in which order) in the table has been indexed.

mode: A "character" indicating whether the database is in read or write mode. Write mode implies read mode.

chrMap: A "character" which is a placeholder, for now.

.tmpFile: A "character". Only for developers..

Details

For all practical purposes, the class may be considered to point to a specific table in an SQLite database. A connection to the database is opened automatically and a pool of connections is maintained.

Methods

ExpData(dbFilename, tablename, mode, indexColumns, pragma) A constructor function. The last three arguments are for expert users.

getDBConnection Returns a connection to the database associated with the ExpData object.

getDBFilename Returns the filename of the database associated with the ExpData object.

`getTablename` Returns the tablename of the ExpData object

`getSchema` Returns the schema of the table associated with the ExpData object.

`getIndexColumns` Returns the indexColumns of the object.

`getColnames` Returns all columns (argument `all = TRUE`) or all columns except the indexColumns (argument `all = FALSE`).

`listTables` Returns all vector of tables in a database.

`getMode` Returns the mode of the ExpData object.

[`signature(x = "ExpData")`: subsetting of the object. ExpData objects do not have rownames.

$ `signature(x = "ExpData")`: selects a column of the table.

head `signature(x = "ExpData")`: prints the first 10 rows of the object.

initialize `signature(.Object = "ExpData")`: The initialize method; use the constructor function `ExpData` instead.

show `signature(object = "ExpData")`: the show method.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

The package vignettes.

Examples

```
showClass("ExpData")
```

`getRegion` *Select a region from an ExpData object.*

Description

This function selects a subset of the data that falls into a particular contiguous genomic region.

Usage

```
getRegion(expData, chr, start, end, strand, what = "*",
  whereClause = "", verbose = getOption("verbose"))
```

Arguments

`expData`	An object of class `ExpData`.
`chr`	Chromosome number of desired region.
`start`	Start position of desired region. If omitted, it is set to 0.
`end`	End position of desired region. If omitted, it is set to 1e12.
`strand`	Strand of desired region. Values of 1 or -1 return data from forward or reverse strand. A value of 0 or a missing argument returns data from any strand, including data with missing strand information.

what	A vector of column names specifying which columns of the data should be returned. Defaults to all columns.
whereClause	Additional filtration criteria, customizable to refer to additional data columns. See Details for more explanation.
verbose	Logical indicating whether details should be printed.

Details

The argument whereClause should be a string indicating a subset of the data to be selected, using SQL syntax. For example, if you have a column called category, you could specify category = 1 to select only those data entries where category has a value of 1. This function operates as a database query, and this argument can include logical combinations of multiple criteria.

Value

Returns a data frame containing the data from the desired region, with the desired columns.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
c1 <- getRegion(ed, chr = 1)
dim(c1)
head(c1)
```

---

importFromAlignedReads

Import aligned reads to database

Description

This function takes a named list of AlignedRead objects (from the **ShortRead** package) and creates an ExpData object from them, with one column for each list element. Column names are taken from list names, which must be unique.

Usage

```
importFromAlignedReads(x, chrMap, dbFilename,
  tablename, overwrite = TRUE, deleteIntermediates = TRUE,
  readPosition = c("5prime", "left", "center"),
  verbose = getOption("verbose"), ...)
```

Arguments

x	This argument can be one of two things: either a named list of objects of class AlignedRead or a named character vector of filenames. In both cases, the names of the object are used as column names inthe resulting database (not that it is not easy to change those names). Therefore the names of x needs to be present and non-empty and also to satisfy the requirements of column names in SQLite. If x is a list of AlignedRead, the column names needs to be unique. If x is a character vector of filenames, the names do not have to be unique, in which case two filenames with the same (column) name gets collapsed into the same column.
chrMap	A vector of chromosome names from the aligned output. On importation to the database, chromosome names will be converted to integers corresponding to position within the chrMap vector.
dbFilename	The filename of the database to which the data will be imported.
tablename	Name of database table to write output data to.
overwrite	Logical indicating whether database table referred to in tablename argument should be overwritten.
deleteIntermediates	Logical indicating whether intermediate database tables constructed in the process should be removed.
readPosition	How each read is assigned a unique genomic location. Default is "5prime" indicating that the location is the position of the 5' end of the reads, "left" indicates that the position of the left part of the read is used (5' end for reads mapping to the forward strand, 3' for reads mapping to the reverse strand), "center" indicates that the position of the center of the read is used.
verbose	Logical indicating whether details should be printed.
...	Additional arguments to be passed to readAligned from **ShortRead**.

Details

The reads are aggregated and joined to form a database where each file/list element is a column. Positions are stored as the position of the 5' end of the reads (note that this differs from the convention for the AlignedRead class from **ShortRead**.) This can be changed by the readPosition argument.

If the x argument is a character vector of filenames, the function will require enough memory to parse each input file in turn. If there are duplicates in names of x the function requires enough memory to parse all files with the same column name at the same time.

If the AlignedRead class object has a weights column in its alignData slot, this weights column is used as the data to aggregate over.

Value

Outputs an object of class ExpData with a column for each element of the x argument.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information. See also ExpData-class, AlignedRead-class and readAligned.

Examples

```
## Not run:
require(ShortRead)
require(yeastRNASeq)
data("yeastAligned")
chrMap <- levels(chromosome(yeastAligned[[1]]))
eData <- importFromAlignedReads(yeastAligned, chrMap = chrMap,
               dbFilename = tempfile(), tablename = "raw",
               overwrite = TRUE)

## End(Not run)
```

---

importToExpData *Import data to database*

Description

This function imports data from a data frame to a table in a database.

Usage

```
importToExpData(df, dbFilename, tablename, overwrite = FALSE,
  verbose = getOption("verbose"), columns = NULL)
```

Arguments

df	A data frame containing data to be imported. Must have columns chr, location and strand.
dbFilename	The filename of the database to which the data will be imported.
tablename	Name of database table to write output data to.
overwrite	Logical indicating whether database table referred to in tablename argument should be overwritten.
verbose	Logical indicating whether details should be printed.
columns	Vector of column names of columns to be imported.

Value

Returns an object of class ExpData.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information. See also ExpData-class.

Examples

```
N  <- 10000 # the number of observations.
df <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                 location = sample(1:1000, size = N, replace = TRUE),
                 strand = sample(c(1L,-1L), size = N, replace = TRUE))
eDataRaw <- importToExpData(df, dbFilename = tempfile(),
                            tablename = "ex_tbl", overwrite = TRUE)
```

---

joinExpData *Merge ExpData objects*

Description

This function merges multiple ExpData object into one in an efficient manner.

Usage

```
joinExpData(expDataList, fields = NULL, tablename = "aggtable",
  overwrite = TRUE, deleteOriginals = FALSE,
  verbose = getOption("verbose"))
```

Arguments

expDataList	List of ExpData objects. Must all be contained in the same database.
fields	A named list whose names correspond to tables of ExpData objects and whose entries indicate the column names to be pulled from each table.
tablename	Name of database table to write output data to.
overwrite	Logical indicating whether database table referred to in tablename argument should be overwritten.
deleteOriginals	Logical indicating whether original database tables in ExpData objects should be deleted.
verbose	Logical indicating whether details should be printed.

Value

An object of class ExpData containing data columns from all the original ExpData objects.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information.

Examples

```
N  <- 10000 # the number of observations.
df1 <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                  location = sample(1:1000, size = N, replace = TRUE),
                  strand = sample(c(1L,-1L), size = N, replace = TRUE))
df2 <- data.frame(chr = sample(1:16, size = N, replace = TRUE),
                  location = sample(1:1000, size = N, replace = TRUE),
                  strand = sample(c(1L,-1L), size = N, replace = TRUE))

eDataRaw1 <- aggregateExpData(importToExpData(df1, dbFilename = "my.db",
                              tablename = "ex_tbl_1", overwrite = TRUE))
eDataRaw2 <- aggregateExpData(importToExpData(df1, dbFilename = "my.db",
                              tablename = "ex_tbl_2", overwrite = TRUE))
jd <- joinExpData(list(eDataRaw1, eDataRaw2), tablename = "combined",
                  fields = list("ex_tbl_1" = c("counts" = "e1"),
                                "ex_tbl_2" = c("counts" = "e2")))
head(jd)
```

`makeGeneRepresentation` *Compute a gene representation from annotation.*

Description

Computing a gene representation from annotation using a variety of methods.

Usage

```
makeGeneRepresentation(annoData, type = c("UIgene", "Ugene", "ROCE",
"background"), gene.id = "ensembl_gene_id", transcript.id = "ensembl_transcript_id", bind.columns
```

Arguments

`annoData`	A data frame which must contain the columns `chr`, `start`, `end` and `strand` which specifies annotation regions of interest, and optionally additional columns.
`type`	The type of gene representation, see details.
`gene.id`	The column in annoData that holds the gene identifiers (only needed for certain types of representation).
`transcript.id`	The column in annoData that holds the transcript identifiers (only needed for certain types of representation).
`bind.columns`	A character vector of column names that will be kept in the return object. It is assumed (but not checked) that these values are constant for all regions in a gene.
`ignoreStrand`	Is strand ignored? Little testing has been done for the value 'TRUE'.
`verbose`	Want verbose output?

Details

A union representation (Ugene) is simply the union of all bases of all transcripts of the gene, with bases belonging to other genes removed.

A union-intersection representation (UIgene) for a gene is defined as bases that are annotated as belonging to all transcripts of the gene, and not to any other gene.

Regions of constant expression (ROCE) are regions where one would assume that the expression is constant. They are best explained by an example: if transcript A goes from 1 to 4 and transcript B goes from 1 to 6 there are two ROCEs, one from 1 to 4 and one from 5 to 6. It is possible to define ROCEs independent of the gene concept, but in its current implementation regions belonging to more than one gene are removed.

Background is essentially the complement of the annotation.

Value

A `data.frame` with rownames and columns chr, strand, start, end, and possibly additional columns.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

Examples

```
data(yeastAnno)
ui <- makeGeneRepresentation(yeastAnno, type = "background")
```

---

mergeWithAnnotation *Combine data with annotation*

Description

This function creates a data frame containing the data and the corresponding annotation information for each data row included in the annotation.

Usage

```
mergeWithAnnotation(expData, annoData, what = "*",
  ignoreStrand = FALSE, splitBy = NULL, verbose = getOption("verbose"))
```

Arguments

`expData`	An object of class `ExpData`.
`annoData`	A data frame which must contain the columns `chr`, `start`, `end` and `strand` which specifies annotation regions of interest.
`what`	Which columns of `expData` to include.
`ignoreStrand`	Logical indicating whether strand should be ignored. If `TRUE`, data from either strand that falls into an annotation region is included.
`splitBy`	Field on which merged data frame should be split before returning.
`verbose`	Logical indicating whether details should be printed.

Details

Generally this function is good for creating a list of data split by some annotation feature, which can then be applied across.

Value

If `splitBy` is `NULL`, returns a data frame containing the data from `expData` that fall into regions defined by `annoData`, and which includes the annotation information, with columns as specified by `what`. If `splitBy` is non-`NULL`, returns a list of data frames with an element for each unique value of `splitBy` field.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See `Genominator` vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
mergeWithAnnotation(ed, yeastAnno[1:5,])
```

---

plot.genominator.coverage

Create coverage plot

Description

S3 method to plot `genominator.coverage` object. Shows coverage as a function of plotting effort.

Usage

```
## S3 method for class 'genominator.coverage'
plot(x, type = "l", col = NULL,
  draw.totals = TRUE, draw.legend = TRUE, legend.location = NULL, ...)
```

Arguments

`x`	An object of class `genominator.coverage`, as returned by `computeCoverage`.
`type`	Plot type. See `plot`.
`col`	Vector of plotting colors.
`draw.totals`	Logical indicating whether totals should be drawn.
`draw.legend`	Logical indicating whether legend should be drawn.
`legend.location`	Vector giving x and y coordinates of legend position.
`...`	Additional arguments for lower-level functions.

Value

This method is used for its side effect.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information. See also computeCoverage.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
a <- computeCoverage(ed, yeastAnno, effort = 2^(5:18),
                     cutoff = function(x, ...) x > 1)
plot(a, lwd = 5, col = "grey")
plot(a, draw.totals = FALSE)
ygroups <- rep(c("mut", "wt"), c(2,2))
b <- computeCoverage(ed, yeastAnno, grups = ygroups,
                     effort = 2^(5:18), cutoff = function(x, ...) x > 1)
plot(b)
b <- computeCoverage(ed, yeastAnno, groups = ygroups,
                     effort = 2^(5:18), cutoff = function(x, ...) x > 3,
                     smooth = function(probs) {
                       probs = probs + min(probs[probs!=0])
                       probs = probs/sum(probs)
                     })
plot(b)
```

---

plot.genominator.goodness.of.fit

Create goodness-of-fit quantile-quantile plot

Description

S3 method to plot genominator.goodness.of.fit object. Creates a quantile-quantile plot of the observed versus theoretical quantiles of goodness-of-fit statistics based on a chi-squared distribution.

Usage

```
## S3 method for class 'genominator.goodness.of.fit'
plot(x, chisq = FALSE, plotCol = TRUE,
  qqline = FALSE, xlab = "theoretical quantiles",
  ylab = "observed quantiles", main, pch = 16, cex = 0.75, ...)
```

Arguments

x	An object of class genominator.goodness.of.fit, as returned by regionGoodnessOfFit.
chisq	Logical indicating whether chi-squared statistics should be plotted (as opposed to p-values from a chi-squared distribution).
plotCol	Logical indicating whether points at extreme quantiles should be colored.
qqline	Logical indicating whether a qqline should be added, this is a line through the 25%- and 75%-quantiles.
xlab	X-axis label for plot.
ylab	Y-axis label for plot.
main	Main label for plot.
pch	Plotting character type for plot.
cex	A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. See par.
...	Additional arguments for lower-level functions, namely plot.

Details

This function constructs a quantile-quantile plot comparing the distribution of observed statistics to either the uniform 0,1 distribution or the appropriate chi-squared distribution. This plotting function provides a tool to assess whether replicate lanes, flow cells, sample preparations, etc. fit the model described in regionGoodnessOfFit.

Value

This method is used for its side effect.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information. See also regionGoodnessOfFit.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
plot(regionGoodnessOfFit(ed, yeastAnno), chisq = TRUE)
```

regionGoodnessOfFit-methods
Calculate goodness-of-fit statistics

Description

A generic method for calculating chi-squared goodness-of-fit statistics (See details). Dispatches on either a `data.frame` or and `ExpData` object.

Usage

```
## S4 method for signature 'data.frame'
regionGoodnessOfFit(obj,
  denominator = colSums(obj),
  groups = rep("A", ncol(obj)))

## S4 method for signature 'ExpData'
regionGoodnessOfFit(obj, annoData,
  groups = rep("A", length(what)),
  what = getColnames(obj, all = FALSE),
  denominator = c("regions", "lanes"),
  verbose = getOption("verbose"))
```

Arguments

`obj`	`data.frame` or `ExpData`
`annoData`	A data.frame of annotation.
`groups`	A factor or character vector describing which are the replicates.
`denominator`	How to scale the columns to take into account sequencing depth.
`what`	Which columns to choose from the database. Default is all data columns.
`verbose`	Whether or not debugging / timing info should be printed.

Details

This function implements the homogenous Poisson model across lanes as described in the article cited below. This model corresponds to common expression parameter across lanes scaled by a lane-specific offset. Goodness of fit to this model across replicates is a good indication of Poisson variation across lanes. Deviation from this is an indication of overdispersion between replicate lanes.

Value

An list containing the statistics and degrees of freedom. See details. Technically, an S3 object with class genominator.goodness.of.fit

Methods

`signature(obj = "ExpData")` Here `obj` represents the results of a call to `summarizeByAnnotation` or a data.frame with columns representing samples and rows representing regions, i.e. genes. Denominator is how we scale each column, therefore it this must be true: `length(denominator) == ncol(obj)`. Finally, groups determines how columns are aggregated across one another, i.e. which columns are replicates.

`signature(obj = "data.frame")` Here `annoData` is an annotation data frame. `groups` is as above. `what` represents the columns to select choose. `denominator` is either the total lane counts, or the lane counts restricted to `annoData`, or a vector of length `length(groups)`

References

James H. Bullard, Elizabeth A. Purdom, Kasper D. Hansen, Steffen Durinck, and Sandrine Dudoit, "Statistical Inference in mRNA-Seq: Exploratory Data Analysis and Differential Expression" (April 2009). U.C. Berkeley Division of Biostatistics Working Paper Series. Working Paper 247. http://www.bepress.com/ucbbiostat/paper247

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
names(regionGoodnessOfFit(ed, yeastAnno))
```

---

splitByAnnotation *Split data into a list by annotation element.*

Description

This function splits the data into a list of matrices, by annotation element.

Usage

```
splitByAnnotation(expData, annoData, what = "*",
  ignoreStrand = FALSE, expand = FALSE,
  addOverStrands = FALSE, verbose = getOption("verbose"))
```

Arguments

`expData`	An object of class `ExpData`.
`annoData`	A data frame which must contain the columns `chr`, `start`, `end` and `strand` which specifies annotation regions of interest.
`what`	Vector of names of columns of `expData` to be included in output.
`ignoreStrand`	Logical indicating whether strand should be ignored. If `TRUE`, data that falls into the annotation region, regardless of strand, is included.
`expand`	Logical indicating whether positions with no data should be included in output. If `TRUE`, lines are added to the output to give a value for each position, even if this value is 0.
`addOverStrands`	Logical indicating whether data should be added across strands. Only applies when `expand` is `TRUE`.
`verbose`	Logical indicating whether details should be printed.

Details

This function retrieves the data contained in the regions of the annoData object. The return object may be significant in size.

Value

Returns a list of length equal to the number of annotation entries split upon. Each list element is either a matrix of data, or a list with data matrices for each strand included (if expand is TRUE and addOverStrands is FALSE).

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

See Also

See Genominator vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
splitByAnnotation(ed, yeastAnno[1:30,])
```

---

summarizeByAnnotation *Summarize data based on genome annotation.*

Description

This function creates a summarization of columns of the data using specified SQLite functions, applying these summarization function to regions defined in an annotation data frame.

Usage

```
summarizeByAnnotation(expData, annoData,
  what = getColnames(expData, all = FALSE), fxs = c("TOTAL"),
  groupBy = NULL, splitBy = NULL, ignoreStrand = FALSE, bindAnno = FALSE,
  preserveColnames = TRUE, verbose = getOption("verbose"))
```

Arguments

expData	An object of class ExpData.
annoData	A data frame which must contain the columns chr, start, end and strand which specifies annotation regions of interest.
what	Vector of names of data columns to be summarized.
fxs	Vector of strings giving the names of SQLite functions to call on the data column(s).
groupBy	Character vector refering to a column in annoData. Regions will be aggregated over distinct values of this column. Setting this argument will set bindAnno to TRUE. If splitBy is set, meta.id will override.

splitBy	String indicating column of annoData object on which to split results.
ignoreStrand	Logical indicating whether strand should be taken into account in aggregation. If TRUE strand will be ignored.
bindAnno	Logical indicating whether annotation information should be included in the output.
preserveColnames	Logical indicating whether column names should be preserved. Only possible when a single function is being applied.
verbose	Logical indicating whether details should be printed.

Details

Most of the computation is done using SQLite. Depending on the use case, this approach may be significantly faster and use much less memory than the alternative: use splitByAnnotation to retrieve a list with all the data and then use R to summarize over each element of the list. It is (naturally) constrained to the use of operations expressible in (SQLite) SQL.

If meta.id is set to a column in annoData, all regions with the same value of the meta.id will be joined together; a standard use case is labelleing exons of a gene.

Value

If splitBy is not specified, returns a data frame containing results of aggregation functions performed on each region defined in annoData. If splitBy is specified, returns a list of data frames with one entry for each unique value of the column which was split on.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

References

The SQLite website http://www.sqlite.org/lang_aggfunc.html has details on what mathematical functions are implemented.

See Also

See Genominator vignette for more information, as well as the ExpData-class.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
data("yeastAnno")
summarizeByAnnotation(ed, yeastAnno[1:50,])
```

summarizeExpData *Summarize a data column*

Description

This function returns a summary of one or more data columns, as indicated by a particular SQLite query function.

Usage

```
summarizeExpData(expData, what = getColnames(expData, all = FALSE),
  fxs = c("TOTAL"), preserveColnames = TRUE, whereClause = "",
  verbose = getOption("verbose"))
```

Arguments

expData	An object of class ExpData.
what	Vector of names of data columns to be summarized.
fxs	Vector of strings giving the names of SQLite functions to call on the data column.
preserveColnames	Logical indicating whether column names should be preserved.
whereClause	Additional filtration criteria, customizable to refer to additional data columns. See Details for more explanation.
verbose	Logical indicating whether details should be printed.

Details

The argument whereClause should be a string indicating a subset of the data to be selected. For example, if you have a column called category, you could specify "category = 1" to select only those data entries where category has a value of 1. This function operates as a database query, and thus the argument can include logical combinations of multiple criteria using SQL boolean operators.

Value

A vector with results of summarization.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@jhsph.edu>

References

The available SQLite functions are listed here: http://www.sqlite.org/lang_aggfunc.html

See Also

See Genominator vignette for more information.

Examples

```
ed <- ExpData(system.file(package = "Genominator", "sample.db"),
              tablename = "raw")
summarizeExpData(ed)
summarizeExpData(ed, fxs = c("MIN", "MAX", "AVG"))
```

`validAnnotation` *Check for validity of a annotation object.*

Description

Checks whether a data.frame satisfy the requirements for an annotation object.

Usage

```
validAnnotation(annoData)
```

Arguments

`annoData`	A `data.frame`.

Value

This function throws an error if the data.frame is not valid.

Author(s)

James Bullard <bullard@berkeley.edu>, Kasper Daniel Hansen <khansen@berkeley.edu>

See Also

The Genominator user guide.

Examples

```
data(yeastAnno)
validAnnotation(yeastAnno)
```

yeastAnno *Example datasets from Genominator*

Description

3 example datasets from Genominator; 2 contain annotation information from yeast and 1 contain actual data from yeast as well. A bigger dataset is available in the experimental data package yeastRNASeq.

Usage

```
data(yeastAnno)
data(yeastAnno.sources)
data(chr1_yeast)
```

Format

yeastAnno is a data frame with 7124 observations on the following 5 variables: chr, start, end, strand, gene_biotype.

yeastAnno.sources is a list with four components names ensembl.gene, ensembl.transcript, ucsc.sgdGene, ucsc.ensGene containing annotation on yeast from 2 different sources (Ensembl and UCSC), each sources has two different queries (one gene-level, one transcript-level). The annotation was obtained in January 2010 and should not be used for analysis.

chr1_yeast is a data frame containing mock data in yeast from two different samples (labelled mRNA_1 and mRNA_2), linked to distinct genomic locations. There may be several data values linked to each genomic location.

Source

Ensembl and UCSC January 2010.

See Also

There is a discussion of the yeastAnno.sources in the withShortRead vignette.

Examples

```
data(yeastAnno)
head(yeastAnno)
data(yeastAnno.sources)
names(yeastAnno.sources)
head(yeastAnno.sources$ensembl.gene)
data(chr1_yeast)
head(chr1_yeast)
```

Index