Package 'Repitools'

May 9, 2024

Version 1.51.0

Date 2021-11-21

Title Epigenomic tools

Author Mark Robinson <mark.robinson@mls.uzh.ch>, Dario Strbenac <dario.strbenac@sydney.edu.au>, Aaron Statham <a.statham@garvan.org.au>, Andrea Riebler <andrea.riebler@math.ntnu.no>

Maintainer Mark Robinson <mark.robinson@mls.uzh.ch>

LazyLoad Yes

Depends R (>= 3.5.0), methods, BiocGenerics (>= 0.8.0)

Imports parallel, S4Vectors (>= 0.17.25), IRanges (>= 2.13.12), GenomeInfoDb, GenomicRanges, Biostrings, Rsamtools, GenomicAlignments, rtracklayer, BSgenome (>= 1.47.3), gplots, grid, MASS, gsmoothr, edgeR (>= 3.4.0), DNAcopy, Rsolnp, cluster

Suggests ShortRead, BSgenome.Hsapiens.UCSC.hg18

Description Tools for the analysis of enrichment-based epigenomic data. Features include summarization and visualization of epigenomic data across promoters according to gene expression context, finding regions of differential methylation/binding, BayMeth for quantifying methylation etc.

Collate classes.R multiHeatmap.R BAM2GRanges.R FastQC-class.R plotClusters.R annoDF2GR.R GCbiasPlots.R featureScores.R profilePlots.R findClusters.R mergeReplicates.R processNimblegenArrays.R regionStats.R cpgDensityPlot.R featureBlocks.R getProbePositionsDf.R genomeBlocks.R mappabilityCalc.R ChromaBlocks.R writeWig.R abcdDNA.R makeWindowLookup.R sequenceCalc.R genQC.R annoGR2DF.R gcContentCalc.R GCadjustCopy.R enrichmentPlot.R cpgBoxplots.R utils.R absoluteCN.R annotationLookup.R cpgDensityCalc.R blocksStats.R binPlots.R chromosomeCNplots.R checkProbes.R relativeCN.R enrichmentCalc.R clusterPlots.R summarizeScores.R determineOffset.R empBayes.R methylEst.R hyper.R maskOut.R

License LGPL (>= 2)

biocViews DNAMethylation, GeneExpression, MethylSeq

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

git_branch devel

git_last_commit 62a6cd3

git_last_commit_date 2024-04-30

Repository Bioconductor 3.20

Date/Publication 2024-05-08

Contents

`abcdDNA` *A wrapper for fitting the offset-adjusted ABCD-DNA GLM*

Description

This function performs differential analyses, given a `QdnaData` object with the sample-specific offsets already calculated (i.e. call `getSampleOffsets` before calling `abcdDNA`), a coefficient (or set of coefficients) to test and dispersion(s). In essence, the function is a wrapper for constructing the offset matrix, fitting the generalized linear model and performing a likelihood ratio test.

Usage

```
abcdDNA(obj, coef = ncol(obj$design), dispersion = NULL)
```

Arguments

obj	a QdnaData object
coef	coefficient (or coefficients) of the design matrix to test
dispersion	estimate(s) of dispersion to use for negative binomial testing

Details

This function is simply a wrapper for taking the details in an QdnaData object and perform the differential analyses, adjusting for copy number if specified.

Value

a DGEGLM (see the edgeR package) containing the results of the differential comparison

Author(s)

Mark Robinson

References

http://imlspenticton.uzh.ch/robinson_lab/ABCD-DNA/ABCD-DNA.html

See Also

QdnaData,

Examples

```
# library(Repitools)
# qd <- QdnaData(counts=counts, regions=gb, design=design,
#                cnv.offsets=cn, neutral=(regs=="L=4 P=2"))
# qd <- getSampleOffsets(qd,ref=1)
# plotQdnaByCN(qd,cnv.group=regs,idx.ref=3,idx.sam=2)
# f <- abcdDNA(qd, dispersion=.05, coef=2)
# topTags(f)
```

absoluteCN *Calculate and Segment Absolute Copy Number from Sequencing Counts*

Description

This function uses the GCadjustCopy function to convert a matrix of count data into absolute copy number estimates, then it segments them, and reports the copy number of either the input regions or user-defined regions of interest.

Usage

```
  ## S4 method for signature 'data.frame,matrix,GCAdjustParams'
absoluteCN(input.windows, input.counts, gc.params, ...)
  ## S4 method for signature 'GRanges,matrix,GCAdjustParams'
absoluteCN(input.windows, input.counts, gc.params,
                                        segment.sqrt = TRUE, ..., verbose = TRUE)
```

Arguments

input.windows	A data.frame with (at least) columns chr, start, and end, or a GRanges object.
input.counts	A matrix of counts. Rows are genomic windows and columns are samples.
gc.params	A GCAdjustParams object, holding parameters related to mappability and GC content correction of read counts.
segment.sqrt	Whether to square root the absolute copy number estimates before running the segmentation.
...	For the data.frame method; the verbose variable and any additional parameters to pass to the segment function. For the GRanges method; additional parameters for the segmentation.
verbose	Whether to print the progess of processing.

Details

For details of the absolute copy number estimation step, see the documentation for GCadjustCopy.

For details of the segmentation, see segment documentation. By default, no weights are used.

Value

A CopyEstimate object. If regions was not provided, it describes the input windows, otherwise it describes the windows specified by regions.

Author(s)

Dario Strbenac

Examples

```
## Not run:
  library(BSgenome.Hsapiens.UCSC.hg18)
  library(BSgenome.Hsapiens36bp.UCSC.hg18mappability)
  load("inputsReads.RData")
  windows <- genomeBlocks(Hsapiens, chrs = paste("chr", c(1:22, 'X', 'Y'), sep = ''),
                          width = 20000)
  counts <- annotationBlocksCounts(inputsReads, anno = windows, seq.len = 300)

  gc.par <- GCAdjustParams(genome = Hsapiens, mappability = Hsapiens36bp,
                           min.mappability = 50, n.bins = 10, min.bin.size = 10,
                           poly.degree = 4, ploidy = c(2, 4))
 abs.cn <- absoluteCN(input.windows = windows, input.counts = counts, gc.params = gc.par)

## End(Not run)
```

`AdjustedCopyEstimate` *Container for results of GC adjusted copy number estimation.*

Description

Contains the genomic coordinates of regions, the raw counts before GC adjustment, the GC content and mappability of each region, and the polynomial model fit, and the GC-adjusted copy number estimates.

Constructor

`AdjustedCopyEstimate(ploidy, windows, mappability, gc, unadj.CN, models, adj.CN)` Creates a AdjustedCopyEstimate object.

`ploidy` Sets of chromosomes in each sample.

`windows` A `GRanges` object.

`mappability` A numeric vector of mappability. Elements between 0 and 1.

`gc` A numeric vector of GC content Elements between 0 and 1.

`unadj.CN` A matrix of estimated copy numbers after mappability adjustment, but before GC content adjustment, if slot `type` is `"absolute"`. Otherwise, fold changes.

`models` The polynomial models that were fit to the counts.

`adj.CN` A matrix of estimated copy numbers after mappability adjustment and GC content adjustment, if slot `type` is `"absolute"`. Otherwise, a matrix of fold changes, based on GC adjusted absolute copy estimates.

Note that `mappability` and `gc` become metadata columns of `windows` when the object is created.

Superclass

This class inherits from `CopyEstimate`.

Additional Slots

These are added to by absoluteCN or relativeCN

A GRangesList of copy number segmentations for each sample.

unadj.CN, adj.CN.seg A GRangesList of copy number segmentations for each sample, using GC adjusted data.

type A flag that contains if the copy number data is absolute or relative.

AffymetrixCdfFile *Placeholder For AffymetrixCdfFile Documentation*

Description

The documentation is available by typing ?aroma.affymetrix::AffymetrixCdfFile, but to avoid a check warning in the Repitools package, this help file is present.

AffymetrixCelSet *Placeholder For AffymetrixCelSet Documentation*

Description

The documentation is available by typing ?aroma.affymetrix::AffymetrixCelSet, but to avoid a check warning in the Repitools package, this help file is present.

---

annoDF2GR *Convert a* data.frame *to a* GRanges.

Description

Checks that the data.frame has the required columns, chr, start, end, then creates a GRanges, keeping all of the additional columns.

Usage

```
  ## S4 method for signature 'data.frame'
annoDF2GR(anno)
```

Arguments

anno	An data.frame, describing some genomic features.

Details

Extra columns are added to the elementMetadata of the GRanges object.

Value

A GRanges of the annotation.

Author(s)

Dario Strbenac

Examples

```
df <- data.frame(chr = c("chr1", "chr3", "chr7", "chr22"),
                 start = seq(1000, 4000, 1000),
                 end = seq(1500, 4500, 1000),
                 t = c(3.11, 0.93, 2.28, -0.18),
                 gc = c("High", "High", "Low", "High"))

annoDF2GR(df)
```

---

annoGR2DF *Convert an annotated* GRanges *to a* data.frame.

Description

Converting a GRanges that might be annotated with some kind of results to a data.frame is useful, because it allows easier writing to file and viewing in other programs, like a spreadsheet program.

Usage

```
  ## S4 method for signature 'GRanges'
annoGR2DF(anno)
```

Arguments

anno	A GRanges, describing some genomic features.

Details

The column name seqnames is changed to chr, and if all the strands are *, then the strand column is dropped.

Value

A data.frame of the annotation.

Author(s)

Dario Strbenac

Examples

```
require(GenomicRanges)
chrs <- c("chr1", "chr3", "chr7", "chr22")
starts <- seq(1000, 4000, 1000)
ends <- seq(1500, 4500, 1000)
t <- c(3.11, 0.93, 2.28, -0.18)
gc <- c("High", "High", "Low", "High")
gr <- GRanges(chrs, IRanges(starts, ends), strand = '*', t, gc)

annoGR2DF(gr)
```

`annotationBlocksCounts` *Counts the number of sequencing reads within supplied genomic blocks.*

Description

Counts reads inside blocks.

Usage

```
  ## S4 method for signature 'ANY,data.frame'
annotationBlocksCounts(x, anno, ...)
  ## S4 method for signature 'character,GRanges'
annotationBlocksCounts(x, anno, ...)
  ## S4 method for signature 'GRanges,GRanges'
annotationBlocksCounts(x, anno, seq.len = NULL, verbose = TRUE)
  ## S4 method for signature 'GRangesList,GRanges'
annotationBlocksCounts(x, anno, ...)
```

Arguments

`x`	A character vector of BAM paths, a `GRangesList`, or `GRanges` object.
`anno`	A set of genomic features to make windows around a reference point of theirs. Either a `data.frame` with (at least) colums `chr`, `start`, and `end`, or a `GRanges` object.
`seq.len`	If sequencing reads need to be extended, the fragment size to be used. Default: NULL (no extension).
`verbose`	Whether to print progress. Default: TRUE.
`...`	Parameters described above, that are not used in the top-level error-checking stage, but are passed further into a private function that uses them in its processing.

Value

A `matrix` of counts is returned, one column per sample and one row per row of genomic features supplied.

Author(s)

Aaron Statham

See Also

annotationCounts, genomeBlocks

Examples

```
require(GenomicRanges)
reads <- GRanges(seqnames = rep("chr1", 5),
                 IRanges(c(3309, 4756, 4801, 4804, 5392), width = 36),
                 strand = c('+', '-', '-', '+', '+'))
genes <- GRanges("chr1", IRanges(5000, 7000), strand = '+')
annotationBlocksCounts(reads, genes, 300)
```

---

annotationBlocksLookup

Forms a mapping between probe locations and chromosomal blocks (regions).

Description

Starting from a table of genome locations for probes, and a table of regions of interest, this procedure forms a list structure that contains the indices to map from one to the other.

Usage

```
  ## S4 method for signature 'data.frame,data.frame'
annotationBlocksLookup(x, anno, ...)
  ## S4 method for signature 'data.frame,GRanges'
annotationBlocksLookup(x, anno, verbose = TRUE)
```

Arguments

x	probe genomic locations, a data.frame with required elements chr, position, and optionally index
anno	a data.frame with required elements chr, start, end, strand and optional element name. Also may be a GRanges with optional elementMetadata column name.
verbose	Whether to print progress to screen.
...	Represents the verbose parameter, when the data.frame,data.frame method is called.

Details

Strandedness of probes is ignored, even if it is given.

If x has no index column, then the probes are given indices from 1 to the number of probes, in the order that they appear in the data.frame or GRanges object.

Value

A list with elements

indexes	a list for each gene in y, giving a vector of indices to the probe data.
offsets	a list for each gebe in y, giving a vector (corresponding to indexes) of offsets relative to the start of the block.

Author(s)

Aaron Statham, Mark Robinson

See Also

annotationLookup which simplifies annotation lookups for constant sized regions

Examples

```
# create example set of probes and gene start sites
probeTab <- data.frame(position=seq(1000,3000,by=200), chr="chrX", strand="+")
genes <- data.frame(chr="chrX", start=c(2100,2200), end=c(2500, 2400), strand=c("+","-"))
rownames(genes) <- paste("gene",1:2,sep="")

# Call annotationLookup() and look at output
annotationBlocksLookup(probeTab, genes)
```

---

annotationCounts *Counts the number of sequencing reads surrounding supplied annotations*

Description

Counts are made in windows with boundaries fixed distances either side of a reference point.

Usage

```
# ANY,data.frame method
annotationCounts(x, anno, ...)
# ANY,GRanges method
annotationCounts(x, anno, up, down, ...)
```

Arguments

x: A character vector of BAM paths, `GRangesList`, or `GRanges` object.

anno: A set of genomic features to make windows around a reference point of theirs. Either a `data.frame` with (at least) colums `chr`, `start`, and `end`, or a `GRanges` object.

up: The number of bases upstream to look.

down: The number of bases downstream to look.

seq.len: If sequencing reads need to be extended, the fragment size to be used. Default: NULL (no extension).

verbose: Whether to print progress. Default: TRUE.

...: Parameters described above, that are not used in the function called, but are passed into annotationBlocksCounts, that uses them in its processing.

Details

If the genomic features annotation contains all unstranded features, the up and down distances refer to how far towards the start of a chromosome, and how far towards the end to make the counting window boundaries. If the annotation is all stranded, then the up and down distances are relative to the TSS of the features.

Value

A `matrix` of counts is returned, one column per sample and one row per row of genomic features supplied.

Author(s)

Aaron Statham

See Also

annotationBlocksCounts, genomeBlocks

Examples

```
require(GenomicRanges)
reads <- GRanges(seqnames = rep("chr1", 5),
                 IRanges(c(3309, 4756, 4801, 4804, 5392), width = 36),
                 strand = c('+', '-', '-', '+', '+'))
genes <- GRanges("chr1", IRanges(5000, 7000), strand = '+')

annotationCounts(reads, genes, 500, 500, 300)
```

annotationLookup *Forms a mapping between probes on a tiling array and windows surrounding the TSSs of genes.*

Description

Starting from genome locations for probes and a locations for a set of genes, this procedure forms a list structure that contains the indices to map from one to the other.

Usage

The data.frame,data.frame method:
```
annotationLookup(x, anno, ...)
```
The data.frame,GRanges method:
```
annotationLookup(x, anno, up, down, ...)
```

Arguments

x: Probe genomic locations, a `data.frame` with required elements `chr`, `position`, and optionally `index`

anno: a `data.frame` with required elements `chr`, `start`, `end`, `strand` and optional element `name`. Also may be a `GRanges` with optional elementMetadata column `name`.

up: The number of bases upstream to look.

down: The number of bases downstream to look.

verbose: Whether to print progress to screen. Default: TRUE

...: Parameters described above, that are not used in the function called, but are passed further into annotationBlocksLookup, which uses them in its processing.

Details

This function is a wrapper for the generic function `annotationBlocksLookup` which can handle annotations of varying sizes. `annotationLookup` is appropriate where you wish to map probes that are within a fixed distance of points of annotation e.g gene transcription start sites. Even if strand information is given for probes, it is ignored.

If `x` has no index column, then the probes are given indices from 1 to the number of probes, in the order that they appear in the `data.frame` or `GRanges` object.

It is an error for the gene annotation to have unstranded features.

Value

A list with elements

a list for each gene in `y`, giving a vector of indices to the probe data.

indexes **offsets** a list for each gebe in `y`, giving a vector (corresponding to `indexes`) of offsets relative to the genes' TSSs for each probe that mapped that that gene.

Author(s)

Aaron Statham, Mark Robinson

See Also

annotationBlocksLookup, makeWindowLookupTable

Examples

```
# create example set of probes and gene start sites
probes <- data.frame(position=seq(1000, 3000, by = 200), chr = "chrX", strand = '-')
genes <- data.frame(chr = "chrX", start=c(2100, 1000), end = c(3000, 2200),
                    strand=c("+","-"))
rownames(genes) <- paste("gene", 1:2, sep = '')

# Call annotationLookup() and look at output
annotationLookup(probes, genes, 500, 500)
```

BAM2GenomicRanges *Read in a (list of) BAM file(s) into a GRanges(List) object.*

Description

A wrapper script for coverting the contents of BAM files for use with GenomicRanges classes.

Usage

```
  ## S4 method for signature 'character'
BAM2GRanges(path, what = character(),
      flag = scanBamFlag(isUnmappedQuery = FALSE, isDuplicate = FALSE),
      verbose = TRUE)
  ## S4 method for signature 'character'
BAM2GRangesList(paths, what = character(),
      flag = scanBamFlag(isUnmappedQuery = FALSE, isDuplicate = FALSE),
      verbose = TRUE)
```

Arguments

path	A character vector of length 1. The path of the BAM file.
paths	A character vector of possibly any length. The paths of the BAM files.
what	What optional attributes of a read to retain. See scanBam and the value section.
flag	What kinds of reads to retain. See ScanBamParam and the flag argument.
verbose	Whether to print the progess of processing.

Value

For the single pathname method; a GRanges object. For the multiple pathnames method; a GRanges-List object.

Author(s)

Dario Strbenac

Examples

```
tiny.BAM <- system.file("extdata", "ex1.bam", package = "Rsamtools")
if(length(tiny.BAM) > 0)
  print(BAM2GRanges(tiny.BAM))
```

BayMethList *Class* "BayMethList"

Description

This S4 class captures the genomic windows together with the number of read counts obtained by affinity-enrichment sequencing experiments for a fully methylated control and one or more samples of interest. Furthermore CpG-density is stored.

Constructor

Creates a BayMethList object:

`BayMethList(windows, control, sampleInterest, cpgDens, f=matrix(), priorTab=list(), methEst=list(), maskEmpBayes=logical())`

`windows` A GRanges object.

`control` A matrix of read counts obtained by an affinity enrichment sequencing experiment for the fully methylated (SssI) treated sample. The number of rows must be equal to `length(windows)`. Each column contains the counts of one sample. The number of columns must be either one or equal to the number of columns of `sampleInterest`.

`sampleInterest` A matrix of read counts obtained by an affinity enrichment sequencing experiment for the samples of interest. The number of rows must be equal to `length(windows)`. Each column contains the counts of one sample.

`cpgDens` A numeric vector containing the CpG density for `windows`. The length must be equal to `length(windows)`

`fOffset` A matrix where each column contains the normalizing offsets for one sample. The number of rows must be either equal to one or the number of windows.

`priorTab` A list containing for each sample of interest the prior parameters as determined by `empBayes`.

`methEst` A list containing the methylation estimates as determined by `methylEst`.

`maskEmpBayes` A logical vector indicating which bins should be masked out in the empirical Bayes analysis. TRUE indicates to neglect the bin in the empirical Bayes approach.

Methods

x[i] `signature(x = "BayMethList")`: Creates a BayMethList object, keeping only the `i` entries.

length `signature(x= "BayMethList")`: gets the number of genomic regions included.

control<- `signature(x = "BayMethList")`: replace the `control` slot

control `signature(object = "BayMethList")`: extract the `control` matrix slot.

cpgDens<- `signature(x = "BayMethList")`: replace the `cpgDens` slot

cpgDens `signature(object = "BayMethList")`: extract the `cpgDens` slot.

sampleInterest<- `signature(x = "BayMethList")`: replace the `sampleInterest` slot

sampleInterest `signature(object = "BayMethList")`: extract the `sampleInterest` matrix slot.

show `signature(object = "BayMethList")`: show an overview of the object

windows<- `signature(x = "BayMethList")`: replace the `windows` slot

windows `signature(object = "BayMethList")`: extract the `windows` GRanges slot.

fOffset<- `signature(x = "BayMethList")`: replace the `fOffset` slot

fOffset `signature(object = "BayMethList")`: extract the `fOffset` slot.

priorTab<- `signature(x = "BayMethList")`: replace the `priorTab` slot

priorTab `signature(object = "BayMethList")`: extract the `priorTab` slot.

methEst<- `signature(x = "BayMethList")`: replace the `methEst` slot

methEst `signature(object = "BayMethList")`: extract the `methEst` slot.

maskEmpBayes<- `signature(x = "BayMethList")`: replace the `maskEmpBayes` slot

maskEmpBayes `signature(object = "BayMethList")`: extract the `maskEmpBayes` slot.

ncontrol `signature(object = "BayMethList")`: get the number of provided SssI samples.

nsampleInterest `signature(object = "BayMethList")`: get the number of provided samples of Interest.

Author(s)

Andrea Riebler and Mark Robinson

See Also

determineOffset, empBayes, methylEst

Examples

```
if(require(BSgenome.Hsapiens.UCSC.hg18)){
    windows <- genomeBlocks(Hsapiens, chrs="chr21", width=100, spacing=100)
    cpgdens <- cpgDensityCalc(windows, organism=Hsapiens,
        w.function="linear", window=700)
    co <- matrix(rnbinom(length(windows), mu=10, size=2), ncol=1)
    sI <- matrix(rnbinom(2*length(windows), mu=5, size=2), ncol=2)
    bm <- BayMethList(windows=windows, control=co, sampleInterest=sI,
        cpgDens=cpgdens)
```

```
    cat("Number of genomic regions", length(bm), "\n")
    cat("Number of fully methylated control samples:", ncontrol(bm), "\n")
    cat("Number of samples of interest:", nsampleInterest(bm), "\n")
    bm[2:20]
}
```

`binPlots` *Create line plots of averaged signal across a promoter*

Description

Using a specified ordering of genes, they are split into multiple bins. In each bin, the signal across is summarized and displayed visually.

Usage

```
  ## S4 method for signature 'ScoresList'
binPlots(x, summarize = c("mean", "median"), ordering = NULL,
  ord.label = NULL, plot.type = c("line", "heatmap", "terrain"), n.bins = 10, cols = NULL,
   lwd = 3, lty = 1, same.scale = TRUE, symm.scale = FALSE, verbose = TRUE)
```

Arguments

`x`	A `ScoresList` object. See `featureScores`.
`summarize`	How to summarise the scores for each bin into a single value.
`ordering`	A `data.frame` of either numeric or factor variables, with the same number of rows as the annotation used to create `x`, or a vector of such types.
`ord.label`	Character string that describes what type of data the ordering is. e.g. "log2 expression". Used to label relevant plot axis.
`plot.type`	Style of plot to draw.
`n.bins`	The number of bins to split the features into, before summarisation.
`cols`	A vector of colours to use for the bins. In order from the lowest value bin, to the highest value bin.
`lwd`	Line width of lines in line plot (either scalar or vector).
`lty`	Line type of line in line plot (either scalar or vector).
`same.scale`	Whether to keep the scale on all plots be the same.
`symm.scale`	Whether the scale on plots is symmetrical around 0.
`verbose`	Whether to print details of processing.

Details

If `plotType = "line"`, a line is plotted for each bin across the promoter.

If `plotType = "heatmap"`, a series of bins are plotted as a heatmap. This can be useful to display a larger number of bins.

If `plotType = "terrain"`, a series of bins are plotted as a 3D-terrain map. This can be useful to display a larger number of bins.

Value

Either a single- or multiple-panel figure.

Author(s)

Mark Robinson

Examples

```
data(chr21genes)
data(samplesList)  # Loads 'samples.list.subset'.
data(expr)  # Loads 'expr.subset'.

fs <- featureScores(samples.list.subset, chr21genes, up = 5000, down = 1000, dist = "base", freq = 1000,
                    s.width = 500)
fs@scores <- list(tables(fs)[[2]] - tables(fs)[[4]])
names(fs) <- "PC-Norm"

binPlots(fs, ordering = expr.subset, ord.label = "expression", plot.type = "line", n.bins = 4)
binPlots(fs, ordering = expr.subset, ord.label = "expression", plot.type = "heatmap", n.bins = 8)
```

`blocksStats` *Calculate statistics for regions in the genome*

Description

For each region of interest or TSS, this routine interrogates probes or sequence data for either a high level of absolute signal or a change in signal for some specified contrast of interest. Regions can be surroundings of TSSs, or can be user-specified regions. The function determines if the `start` and `end` coordinates of `anno` should be used as regions or as TSSs, if the up and down coordinates are `NULL` or are numbers.

Usage

The ANY,data.frame method:
`blocksStats{ANY,data.frame}(x, anno, ...)`
The ANY,GRanges method:
`blocksStats{ANY,GRanges}(x, anno, up = NULL, down = NULL, ...)`

Arguments

x: A `GRangesList`, `AffymetrixCelSet`, or a `data.frame` of data. Or a `character` vector of BAM paths to the location of the BAM files.

anno: Either a `data.frame` or a `GRanges` giving the gene coordinates or regions of interest. If it is a `data.frame`, then the column names are (at least) `chr`, `name`, `start`, `end`. Column `strand` is also mandatory, if `up` and `down` are `NULL`.

seq.len: If sequencing reads need to be extended, the fragment size to be used.

p.anno: A data.frame with (at least) columns chr, position, and index. This is an optional parameter of the AffymetrixCelSet method, because it can be automatically retrieved for such array data. The parameter is also optional, if mapping is not NULL.

mapping: If a mapping with annotationLookup or annotationBlocksLookup has already been done, it can be passed in, and avoids unnecessary re-conmputing of the mapping list within blocksStats.

chrs: If p.anno is NULL, and is retrieved from an ACP file, this vector gives the textual names of the chromosomes.

log2.adj: Whether to take log_2 of array intensities.

design: A design matrix specifying the contrast to compute (i.e. The samples to use and what differences to take.).

up: The number of bases upstream to consider in calculation of statistics. If not provided, the starts and ends in anno are used as region boundaries.

down: The number of bases upstream to consider in calculation of statistics. If not provided, the starts and ends in anno are used as region boundaries.

lib.size: A string that indicates whether to use the total lane count, total count within regions specified by anno, or normalisation to a reference lane by the negative binomial quantile-to-quantile method, as the library size for each lane. For total lane count use "lane", for region sums use "blocks", and for the normalisation use "ref".

robust: Numeric. If it is 0, then a robust linear model is not fitted. If it is greater than 0, a robust linear model is used, and the number specifies the minimum number of probes a region has to have, for statistics to be reported for that region.

p.adj: The method used to adjust p-values for multiple testing. Possible values are listed in p.adjust.

Acutoff: If libSize is "ref", this argument must be provided. Otherwise, it must not. This parameter is a cutoff on the "A" values to take, before calculating trimmed mean.

verbose: Logical; whether to output commments of the processing.

... Parameters described above, that are not used in the function called, but are passed further into a private function that uses them in its processing.

Details

For array data, the statstics are either determined by a t-test, or a linear model. For sequencing data, the two groups are assumed to be from a negative binomial distribution, and an exact test is used.

Value

A data.frame, with the same number of rows as there are features described by anno, but with additional columns for the statistics calculated at each feature.

Author(s)

Mark Robinson

See Also

annotationLookup and annotationBlocksLookup

Examples

```
require(GenomicRanges)
intensities <- matrix(c(6.8, 6.5, 6.7, 6.7, 6.9,
                        8.8, 9.0, 9.1, 8.0, 8.9), ncol = 2)
colnames(intensities) <- c("Normal", "Cancer")
d.matrix <- matrix(c(-1, 1))
colnames(d.matrix) <- "Cancer-Normal"
probe.anno <- data.frame(chr = rep("chr1", 5),
                         position = c(4000, 5100, 6000, 7000, 8000),
                         index = 1:5)
anno <- GRanges("chr1", IRanges(7500, 10000), '+', name = "Gene 1")
blocksStats(intensities, anno, 2500, 2500, probe.anno, log2.adj = FALSE, design = d.matrix)
```

---

checkProbes *Check Probe Specificity for Some Regions*

Description

Given a set of gene coordinates, and probe mappings to the genome, a plot is created across every gene region of how many probes mapped to each position.

Usage

```
  ## S4 method for signature 'data.frame,data.frame'
checkProbes(regs, probes, up = NULL, down = NULL, ...)
  ## S4 method for signature 'GRanges,GRanges'
checkProbes(regs, probes, up = NULL, down = NULL, ...)
```

Arguments

regs	A data.frame with (at least) columns chr, start, end, strand, and name, or a GRanges object with an elementMetadata column name. The starts and ends of regions describe are the windows plotted in.
probes	A data.frame describing where the probes mapped to, with (at least) columns name (identifier of a probe), chr, start, and end, or a GRanges object with an elementMetadata column name.
up	How many bases upstream to plot.
down	How many bases downstream to plot.
...	Line parameters passed onto matplot.

Details

If up and down are NULL, then the gene is plotted as it is described by its start and end coordinates.

This function produces a number of plots. Sending output to a PDF device is recommended.

Value

A set of plots is created, one for each of the genes. The lines in the plot show where a probe hits (the x - axis) and how many places in total the probe hits in the genome (y - axis).

Author(s)

Dario Strbenac

Examples

```
p.table <- data.frame(name = c("probeA", "probeB", "probeC", "probeC", "probeC"),
    strand = c('+', '-', '+', '-', '-'),
                              chr = c("chr1", "chr2", "chr1", "chr2", "chr2"),
                            start = c(20, 276, 101, 101, 151),
                              end = c(44, 300, 125, 125, 175))
r.table <- data.frame(name = c("gene1", "gene2", "gene3"),
                              chr = c("chr1", "chr2", "chr2"),
                           strand = c('+', '-', '+'),
                            start = c(20, 500, 75),
                              end = c(200, 800, 400))
pdf("tmp.pdf", height = 6, width = 14)
checkProbes(r.table, p.table, lwd = 4, col = "blue")
dev.off()
```

---

chr21genes *Positions of Genes on Human Chromosome 21*

Description

Annotation of chromosome 21 genes from RefSeq in June 2010.

Usage

```
chr21genes
```

Format

A data frame.

Source

UCSC Genome Browser tables.

ChromaBlocks *A function to find areas of enrichment in sequencing data*

Description

This function discovers regions of enrichment in ChIP-seq data, using the method described in Hawkins RD. et al 2010 Cell Stem Cell.

Usage

```
## S4 method for signature 'GRangesList,GRangesList'
ChromaBlocks(rs.ip, rs.input, organism, chrs, ipWidth=100, inputWidth=500, preset=NULL, blockWidth=NUL
```

Arguments

rs.ip	A GRangesList object containing reads from the Immunoprecipited sample. If multiple lanes are supplied, they are pooled.
rs.input	A GRangesList object containing reads from the Input (unenriched) sample. If multiple lanes are supplied, they are pooled.
organism	The BSgenome object
chrs	An character or integer vector with the indicies of the chromosomes of the organism object to analyse
ipWidth	Size in basepairs of the windows to use for the IP samples
inputWidth	Size in basepairs of the windows to use for the Input samples
preset	Either "small", "large" to use cutoffs described in Hawkins et al or NULL (where blockWidth, minBlocks must be specified)
blockWidth	Number of adjacent blocks to consider at once
minBlocks	The minimum number of blocks required above cutoff
extend	Optional: whether to extend significant blocks until adjacent blocks are less than this value
cutoff	Optional: the cutoff to use to call regions. If left as NULL a cutoff will be chosen which satisfied the specified FDR
FDR	The target False Discovery Rate; If cutoff is not supplied, one will be chosen to satisfy this value
nPermutations	The number of permutations of the data to determine the cutoff at the supplied FDR
nCutoffs	The number of different cutoffs to try to satisfy the FDR, a higher value will give finer resolution but longer processing time
cutoffQuantile	The quantile of the RPKM to use as the maximum cutoff tried; a higher value will give lower resolution but may be needed if a cutoff satisfying the FDR cannot be determined with the default value
verbose	logical, whether to output commments of the processing
seq.len	If sequencing reads need to be extended, the fragment size to be used

Value

A ChromaResults object.

Author(s)

Aaron Statham

See Also

ChromaResults

ChromaResults-class *ChromaResults class*

Description

The ChromaResults class stores the results of a ChromaBlocks run.

Slots of a ChromaResults object

blocks: GRanges of the blocks used across the genome, with their calculated RPKM regions: IRangesList of regions determined to be enriched FDRTable: data.frame showing the FDR at each cutoff tested cutoff: The cutoff used to determine enrichment

Author(s)

Aaron Statham

See Also

ChromaBlocks

---

chromosomeCNplots *Plot copy number by chromosome*

Description

Generates plots of position along chromosomes vs. estimated copy number. If GC adjustment was performed, then there are two plots per page; one before adjustment and one after adjustment.

Usage

```
  ## S4 method for signature 'CopyEstimate'
chromosomeCNplots(copy, y.max = NULL, pch = 19, cex = 0.2,
           pch.col = "black", seg.col = "red", lty = 1, lwd = 2, verbose = TRUE)
  ## S4 method for signature 'AdjustedCopyEstimate'
chromosomeCNplots(copy, y.max = NULL, pch = 19, cex = 0.2,
           pch.col = "black", seg.col = "red", lty = 1, lwd = 2, verbose = TRUE)
```

Arguments

copy	A CopyEstimate or AdjustedCopyEstimate object.
y.max	The maximum value of the y-axis of the scatter plots.
pch	Style of points in the scatter plots.
cex	Whether to square root the absolute copy number estimates before running the segmentation.
pch.col	Colour of points in the scatter plots.
seg.col	Colour of copy number segmentation line.
lty	Line type of plotted regression line.
lwd	Line width of plotted regression line.
verbose	Whether to print the progess of processing.

Details

See absoluteCN or relativeCN for how to do the GC adjusted copy number estimates, if this is required. The segmentation line plotted is of the segmentation regions found by circular binary segmentation.

Value

A number of pages of scatterplots. The output should, therefore, be sent to a PDF device.

Author(s)

Dario Strbenac

Examples

```
## Not run:
  library(BSgenome.Hsapiens.UCSC.hg18)
  library(BSgenome.Hsapiens36bp.UCSC.hg18mappability)
  load("inputsReads.RData")
  windows <- genomeBlocks(Hsapiens, chrs = paste("chr", c(1:22, 'X', 'Y'), sep = ''),
                          width = 20000)
  counts <- annotationBlocksCounts(inputsReads, anno = windows, seq.len = 300)

  gc.par <- GCAdjustParams(genome = Hsapiens, mappability = Hsapiens36bp,
                           min.mappability = 50, n.bins = 10, min.bin.size = 10,
                           poly.degree = 4, ploidy = c(2, 4))
 abs.cn <- absoluteCN(input.windows = windows, input.counts = counts, gc.params = gc.par)

  pdf("chrProfiles.pdf")
  chromosomeCNplots(abs.cn, y.max = 8)
  dev.off()

## End(Not run)
```

ClusteredScoresList *Container for coverage matrices with clustering results.*

Description

Contains a list of coverage matrices, the parameters that were used to generate them origin, and also cluster membership and expression data.

It also allows the user to take the ScoresList output of featureScores, and do their own custom clustering on the coverage matrices, then save the clustering results in this container.

Constructor

ClusteredScoresList(x, anno = x@anno, scores = tables(x), expr = NULL, expr.name = NULL, cluster.id, sort.name = NULL, sort.data = NULL) Creates a ClusteredScoresList object.

x A ScoresList object.

anno A GRanges object. Give this value if only a subset of features was used for clustering.

scores A list of coverage matrices. Give this if the matrices in x were modified before clustering.

expr A numeric vector, same length as number of rows of every coverage matrix.

expr.name A label, describing the expression data.

cluster.id A vector, same length as number of rows of every coverage matrix.

sort.data Vector of data to order features within clusters by.

sort.name Human readable description of what the sorting data is of.

Subsetting

In the following code snippets, x is a ClusteredScoresList object.

x[i] Creates a ClusteredScoresList object, keeping only the i matrices.

subsetRows(x, i = NULL) Creates a ClusteredScoresList object, keeping only the i features.

clusters(x) Creates a ClusteredScoresList object, keeping only the i features.

Accessors

In the following code snippets, x is a ClusteredScoresList object.

clusters(x) Get the cluster ID of each feature.

Author(s)

Dario Strbenac

clusterPlots *Visualisation of tables of feature coverages.*

Description

Takes the output of featureScores, or a modified version of it, and plots a heatmaps or lineplots representation of clustered coverages.

Usage

```
  ## S4 method for signature 'ClusteredScoresList'
clusterPlots(
    scores.list, plot.ord = 1:length(scores.list), plot.type = c("heatmap", "line", "by.cluster"),
    heat.bg.col = "black", summarize = c("mean", "median"), symm.scale = FALSE, cols = NULL, t.name = NUL
      verbose = TRUE, ...)
  ## S4 method for signature 'ScoresList'
clusterPlots(scores.list, scale = function(x) x,
   cap.q = 0.95, cap.type = c("sep", "all"), all.mappable = FALSE, n.clusters = NULL,
   plot.ord = 1:length(scores.list), expr = NULL, expr.name = NULL, sort.data = NULL,
    sort.name = NULL, plot.type = c("heatmap", "line", "by.cluster"),
   summarize = c("mean", "median"), cols = NULL, t.name = NULL, verbose = TRUE, ...)
```

Arguments

scores.list	A ScoresList or ClusteredScoresList object.
scale	A function to scale all the coverages by. Default : No scaling.
cap.q	The quantile of coverages above which to make any bigger coverages equal to the quantile.
cap.type	If "sep", then the cap quantile is calculated and applied to each coverage matrix separately. If "all", then one cap quantile is calculated based on all of the matrices combined.
all.mappable	If TRUE, then only features with all measurements not NA will be used.
n.clusters	Number of clusters to find in the coverage data. Required.
plot.ord	Order of the experiment types to plot.
expr	A vector of expression values.
expr.name	A label, describing the expression data.
sort.data	A vector of values to sort the features within a cluster on.
sort.name	Label to place under the sort.data plot.
plot.type	Style of plot to draw.
heat.bg.col	If a heatmap is being drawn, the background colour to plot NA values with.
summarize	How to summarise the score columns of each cluster. Not relevant for heatmap plot.

symm.scale	Whether to make lineplot y-axis or heatmap intensity centred around 0. By default, all plots are not symmetrically ranged.
cols	The colours to use for the lines in the lineplot or intensities in the heatmap.
t.name	Title to use above all the heatmaps or lineplots. Ignored when cluster-wise lineplots are drawn.
verbose	Whether to print the progress of processing.
...	Further graphical paramters passed to plot when heatmap plot is drawn, that influence how the points of the expression and sort data plots will look. If the lineplot is being drawn, parameters to influence the line styles.

Details

A ClusteredScoresList should be created by the user, if they wish to do some custom clustering and normalisation on the coverage matrices. Otherwise, if the user is happy with k-means or PAM clustering, then the ScoresList object as output by featureScores() can be directly used. If called with a ScoresList, then the matrices for each coverage type are joined. Then the function supplied by the scale argument is used to scale the data. Next, each matrix is capped. Then each matrix is divided by its maximum value, so that the Euclidean distance between maximum reads and no reads is the same for each matrix. Lastly, either k-means or PAM clustering is performed to get the cluster membership of each feature. If there are any NAs in the scores, then PAM will be used. Otherwise, k-means is used for speed. Then, a ClusteredScoresList object is created, and used. The clusters are guaranteed to be given IDs in descending order of summarised cluster expression, if it is provided. If called with a ClusteredScoresList, no scaling or capping is done, so it is the user's responsibility to normalise the coverage matrices as they see fit, when creating the ClusteredScoresList object.

If a ClusteredScoresList object is subsetted, the original data range is saved in a private slot, so that if the user wants to plot a subset of the features, such as a certain cluster, for example, the intensity range of the heatmap, or the y-axis range of the lineplot will be the same as before subsetting.

If expression data is given, the summarised expression level of each cluster is calculated, and the clusters are plotted in order of decreasing expression, down the page. Otherwise, they are plotted in ascending order of cluster ID. If a heatmap plot is being drawn, then a heatmap is drawn for every coverage matrix, side-by-side, and a plot of each feature's expression is put alongside the heatmaps, if provided. If additional sort vector was given, the data within clusters are sorted on this vector, then a plot of this data is made as the rightmost graph.

The lineplot style is similar to the heatmap plot, but clusters are summarised. A grid, with as many rows as there are clusters, and as many columns as there are clusters is made, and lineplots showing the summarised scores are made in the grid. Beside the grid, a boxplot of expression is drawn for each cluster, if provided.

For a cluster-wise lineplot, a graph is drawn for each cluster, with the colours being the different coverage types. Because it makes sense that there will be more clusters than there are types of coverage (typically double to triple the number), the plots are not drawn side-by-side, as is the layout for the heatmaps. For this reason, sending the output to a PDF device is necessary. It is recommended to make the width of the PDF device wider than the default. Since the coverage data between different marks is not comparable, this method is inappropriate for visualising a ClusteredScoresList object if it was created by the clusterPlots scoresList method. If the user, however, can come up

with a normalisation method to account for the differences that are apparent between different types (i.e. peaked vs. spread) of marks that makes the coverages meaningfully comparable, they can alter the tables, do their own clustering, and create a `ClusteredScoresList` object with the modified tables.

Value

If called with a `ScoresList`, then a `ClusteredScoresList` is returned. If called with a `ClusteredScoresList`, then nothing is returned.

Author(s)

Dario Strbenac

See Also

`featureScores` for generating coverage matrices.

Examples

```
data(samplesList)  # Loads 'samples.list.subset'.
data(expr)  # Loads 'expr.subset'.
data(chr21genes)

fs <- featureScores(samples.list.subset[1:2], chr21genes, up = 2000, down = 1000,
                    freq = 500, s.width = 500)
clusterPlots(fs, function(x) sqrt(x), n.clusters = 5, expr = as.numeric(expr.subset),
             plot.type = "heatmap", pch = 19, cex = 0.5)
```

CopyEstimate *Container for results of fold change copy number estimation.*

Description

Contains the genomic coordinates of regions, and fold change estimates.

Constructor

`CopyEstimate(windows, unadj.CN, unadj.CN.seg)` Creates a CopyEstimate object.

`windows` A `GRanges` object.

`unadj.CN` A matrix of fold changes.

`unadj.CN.seg` A `GRangesList` object holding the segmentation results.

Additional Slots

These are added to by `absoluteCN` or `relativeCN`

A flag that contains if the copy number data is absolute or relative.

type cpgBoxplots *Boxplots of intensity, binned by Cpg Density*

Description

Either makes a side by side boxplot of two designs, or plots a single boxplot for the difference between the two designs.

Usage

```
## S4 method for signature 'AffymetrixCelSet'
cpgBoxplots(this, samples=c(1,2), subsetChrs="chr[1-5]", gcContent=7:18, calcDiff=FALSE, verbose=FALS
## S4 method for signature 'matrix'
cpgBoxplots(this, ndfTable = NULL, organism, samples=c(1,2), subsetChrs="chr[1-5]", gcContent=7:18, ca
```

Arguments

this	Either an AffymetrixCelSet or a matrix of intensity data.
ndfTable	In the case of Nimblegen data, a data.frame with at least columns chr and sequence. Must be in the same order of rows as the intensity data.
organism	The BSgenome object of the genome build to use for getting DNA sequence surrounding the probes.
samples	Which 2 columns from the data matrix to use.
subsetChrs	Which chromosomes to limit the analysis to.
gcContent	A range of GC content, which only probes that have GC content in the range are used for the graphing.
calcDiff	Boolean. Plot the difference between the two samples ?
verbose	Boolean. Print processing output.
nBins	Bins to bin the intensities into.
pdfFile	Name of file to output plots to.
ylim	Y limit of graphs
col	Colour of boxes.
mfrow	Not specified by the user. Rows and columns to draw the plots in.

Details

CpG content of probes is calculated in a 600 base window surrounding the probe, with a linearly decresasing weighting further away from the probe.

Value

Invisibly returns a list of the plots.

Author(s)

Mark Robinson, Dario Strbenac

cpgDensityCalc *Calculate CpG Density in a Window*

Description

Function to calculate CpG density around a position.

Usage

```
  ## S4 method for signature 'data.frame,BSgenome'
cpgDensityCalc(x, organism, ...)
  ## S4 method for signature 'GRangesList,BSgenome'
cpgDensityCalc(x, organism, verbose = TRUE, ...)
  ## S4 method for signature 'GRanges,BSgenome'
cpgDensityCalc(x, organism, seq.len = NULL, window = NULL,
                                  w.function = c("none", "linear", "exp", "log"),
                                           verbose = TRUE)
```

Arguments

x	A data.frame, with columns chr and position, or columns chr, start, end, and strand. Also may be a GRangesList object, or GRanges.
window	Bases around the locations that are in the window. Calculation will consider window/2 - 1 bases upstream, and window/2 bases downstream.
w.function	Weighting function to use. Can be "none", "linear", "log", or "exp"
organism	The BSgenome object to calculate CpG density upon.
seq.len	The fragment size of the sequence reads in x. Default: No extension.
verbose	Print details of processing.
...	Arguments passed into the data.frame or GRangesList method, but not used until the GRanges method.

Details

If the version of the data frame with the start, end, and strand columns is given, the window will be created around the TSS.

For weighting scheme "none", this is equivalent to the number of CG matches in the region. For "linear" weighting, each match is given a score 1/x where x is the number of bases from the postition that the match occurred, and the scores are summed. For exponential weighting and logarithmic weighting, the idea is similar, but the scores decay exponentially (exp^-5x/window) and logarithmically (log2(2 - (distancesForRegion / window))).

Value

A numeric vector of CpG densities for each region.

Author(s)

Dario Strbenac

Examples

```
if(require(BSgenome.Hsapiens.UCSC.hg18))
{
  TSSTable <- data.frame(chr = c("chr1", "chr2"), position = c(100000, 200000))
  cpgDensityCalc(TSSTable, organism = Hsapiens, window = 600)
}
```

---

cpgDensityPlot *Plot the distribution of sequencing reads CpG densities.*

Description

Function to generate a plot of the distribution of sequencing reads CpG densities.

Usage

```
  ## S4 method for signature 'GRangesList'
cpgDensityPlot(x, cols=rainbow(length(x)), xlim=c(0,20), lty = 1, lwd = 1, main="CpG Density Plot", ver
```

Arguments

x	A GRangesList object of reads to plot CpG density of
cols	The line colour for each element of x
xlim	xlim parameter passed to plot.
lty	The line type for each element of x
lwd	The line width for each element of x
main	main parameter passed to plot
verbose	Print details of processing.
...	Arguments passed into cpgDensityCalc. seq.len and organism are required.

Details

See cpgDensityCalc for details of options for calculating the CpG density.

Value

A plot is created. The data processed by cpgDensityCalc is invisibly returned.

Author(s)

Aaron Statham

Examples

```
if(require(BSgenome.Hsapiens.UCSC.hg18))
{
  data(samplesList) # Loads 'samples.list.subset'.
 cpgDensityPlot(samples.list.subset, seq.len=300, organism=Hsapiens, lwd=4, verbose=TRUE)
}
```

`determineOffset` *Function to determine the normalising offset f that accounts for the relative sequencing depth.*

Description

The composition of a library influences the resulting read densities. To adjust the modelled mean (in the Poisson model) for these composition effects, we estimate a normalising factor f that accounts simultaneously for overall sequencing depth and composition. The derivation of this offset is motivated by the M (log ratio) versus A (average-log-count) plot.

Usage

```
determineOffset(x, quantile = 0.998, controlPlot = list(show = FALSE,
    nsamp = 50000, mfrow=c(1,1), xlim=NULL, ylim=NULL, main=NULL, ask=FALSE))
```

Arguments

`x`	`BayMethList` object.
`quantile`	quantile q to restrict values of A = log2(sampleInterest*control)/2
`controlPlot`	list defining whether a MA plot should be shown. - `show` logical. If 'TRUE' the corresponding MA plot is shown. (default FALSE) - `nsamp` number of genomic regions included in the plot. (These are sampled without replacement). - `mfrow` vector of the form "c(nr, nc)" to determine how several plots should be ordered. - `xlim, ylim` numeric vectors of length 2, giving the x and y coordinates ranges. - `main` If NULL the names of the sample of interest are used as title in the MA plot. Alternatively, a vector with length equal to the number of samples of interest can be provided. - `ask` logical. If 'TRUE' (and the R session is interactive) the user is asked for input, before a new figure is drawn. (default FALSE).

Value

A `BayMethList` object given as input, where the slot `fOffset` is filled accordingly.

Author(s)

Andrea Riebler

See Also

maPlot, plotSmear

Examples

```
if(require(BSgenome.Hsapiens.UCSC.hg18)){
     windows <- genomeBlocks(Hsapiens, chrs="chr21", width=100, spacing=100)
     cpgdens <- cpgDensityCalc(windows, organism=Hsapiens,
         w.function="linear", window=700)
     co <- matrix(rnbinom(length(windows), mu=10, size=2), ncol=1)
     sI <- matrix(rnbinom(2*length(windows), mu=5, size=2), ncol=2)
     bm <- BayMethList(windows=windows, control=co, sampleInterest=sI,
         cpgDens=cpgdens)

     bm <- determineOffset(bm, controlPlot=list(show=TRUE, mfrow=c(1,2)))
}
```

---

empBayes *Function to calculate prior parameters using empirical Bayes.*

Description

Under the empirical Bayes approach (and assuming a uniform prior for the methylation level) the shape and scale parameters for the gamma prior of the region-specific read density are derived. The parameters are thereby determined in a CpG-dependent manner.

Usage

```
empBayes(x, ngroups = 100, ncomp = 1, maxBins=50000, method="beta", controlMethod=list(mode="full", wei
```

Arguments

x	Object of class BayMethList.
ngroups	Number of CpG density groups you would like to consider. The bins are classified based on its CpG density into one of ngroups classes and for each class separately the set of prior parameters will be determined.
ncomp	Number of components of beta distributions in the prior distribution for the methylation level when method is equal to beta.
maxBins	Maximum number of bins in one CpG density group used to derive the parameter estimates. If maxBins is smaller than the number of bins that are in one groups than maxBins bins are sampled with replacement.
method	Either DBD for a Dirac-Beta-Dirac mixture, representing a mixture a mixture of a point mass at zero, a beta distribution and a point mass at one, or beta for as Beta mixture with ncomp components.
controlMethod	list defining settings if the Dirac-Beta-Dirac mixture is chosen.

- mode Either full, fixedWeights or fixedBeta. Using the full both the mixture weights and beta parameters are estimated. In mode fixedWeights the weights are fixed given to the values in weights and only the parameters of the beta component are estimated. In mode fixedBeta the parameters of the beta component are fixed to the values specified in param. The default mode is full.
- weights Numeric vector of length three specifying the weights for the Dirac-Beta-Dirac mixture when mode is equal to fixedWeights. The first element specifies the weight for the zero point mass, the second for the beta component and the third for the point mass at one. The three values must sum up to one. The default is c(0.1, 0.8, 0.1).
- param Numeric vector of length two specifying (positive) parameters of the beta distribution component when mode is equal to fixedBeta. The default is c(1,1).

ncpu — Number of CPUs on your machine you would like to use in parallel. If ncpu is set to NULL, half of the CPUs will be used on machines with a maximum of four CPUs, and 2/3 will be used if more CPUs are available.

verbose — Boolean indicating whether the empirical Bayes function should run in a verbose mode (default 'FALSE').

Details

BayMeth takes advantage of the relationship between CpG-density and read depth to formulate a CpG-density-dependent gamma prior distribution for the region-specific read density. Taking CpG-density into account the prior should stabilise the methylation estimation procedure for low counts and in the presence of sampling variability. The shape and scale parameter of the gamma prior distribution are determined in a CpG-density-dependent manner using empirical Bayes. For each genomic bin the CpG density is provided in the BayMethList-object. Each bin is classified based on its CpG-density into one of ngroups non-overlapping CpG-density intervals. For each class separately, we derive the values for the shape and scale parameter under an empirical Bayes framework using maximum likelihood. For CpG classes which contain more than maxBins bins, a random sample drawn with replacement of size maxBins is used to derive these prior parameters. Note that both read depths, from the SssI control and the sample of interest, are thereby taken into account. We end up with ngroups parameter sets for shape and rate.

Value

A BayMethList object where the slot priorTab is filled. priorTab represent a list. The first list entry contains the CpG group a bin is assigned to. The second entry contains the number of components that have been used for the prior (at the moment 1). The following list entries correspond to one sample of interest, respectively, and contain a matrix with the optimal shape and scale parameters for all CpG classes. The first row contains the optimal shape parameter and the second row the optimal scale parameter. The number of columns corresponds to the number of CpG classes specified in ngroups.

Author(s)

Andrea Riebler

Examples

```
if(require(BSgenome.Hsapiens.UCSC.hg18)){
    windows <- genomeBlocks(Hsapiens, chrs="chr21", width=100, spacing=100)
    cpgdens <- cpgDensityCalc(windows, organism=Hsapiens,
        w.function="linear", window=700)
    co <- matrix(rnbinom(length(windows), mu=10, size=2), ncol=1)
    sI <- matrix(rnbinom(2*length(windows), mu=5, size=2), ncol=2)
    bm <- BayMethList(windows=windows, control=co,
        sampleInterest=sI, cpgDens=cpgdens)
    bm <- determineOffset(bm)

    # mask out unannotated high copy number regions
    # see Pickrell et al. (2011), Bioinformatics 27: 2144-2146.

    # should take about 3 minutes for both sample of interests with 2 CPUs.
    bm <- empBayes(bm, ngroups=20)
}
```

---

enrichmentCalc *Calculate sequencing enrichment*

Description

Function to calculate enrichment over the whole genome of sequencing reads.

Usage

```
  ## S4 method for signature 'GRanges'
enrichmentCalc(x, seq.len = NULL, verbose = TRUE)
  ## S4 method for signature 'GRangesList'
enrichmentCalc(x, verbose = TRUE, ...)
```

Arguments

x	A GRangesList or GRanges object. All chromosome lengths must be stored in the Seqinfo of this object.
seq.len	If sequencing reads need to be extended, the fragment size to be used.
verbose	Whether to print the progress of processing.
...	Argument seq.len above, not directly used in the GRangesList method.

Details

If seq.len is supplied, x is firstly extended, and then turned into a coverage object. The number of extended reads covering each base pair of the genome is then tabulated, and returned as a data.frame.

Value

For the GRanges method, data.frame containing columns coverage and bases. For the GRangesList method, a list of such data.frames.

Author(s)

Aaron Statham

Examples

```
require(GenomicRanges)
data(samplesList)  # Loads 'samples.list.subset'.
seqlengths(samples.list.subset)

tc <- enrichmentCalc(samples.list.subset, seq.len = 300)
```

---

enrichmentPlot *Plot the distribution of sequencing enrichment.*

Description

Function to generate a plot of the distribution of sequencing reads enrichments.

Usage

```
  ## S4 method for signature 'GRangesList'
enrichmentPlot(x, seq.len, cols = rainbow(length(x)),
    xlim = c(0, 20), main = "Enrichment Plot", total.lib.size = TRUE, verbose = TRUE, ...)
```

Arguments

x	A GRangesList object of reads to plot enrichment of. The chromosome lengths must be stored in the Seqinfo of this object.
seq.len	The fragment size to be used for extending the sequencing reads.
cols	The line colour for each element of x
xlim	xlim parameter passed to plot, the default is appropriate for "linear" cpgDensityCalc weighting.
main	main parameter passed to plot
total.lib.size	Whether to normalise enrichment values to the total number of reads per lane.
verbose	Print details of processing.
...	Additional graphical parameters to pass to plot.

Details

See enrichmentCalc for details of how the results are determined.

Value

A plot is created. The data processed by `enrichmentCalc` is invisibly returned.

Author(s)

Aaron Statham

Examples

```
data(samplesList)  # GRangesList of reads 'samples.list.subset'
enrichmentPlot(samples.list.subset, seq.len = 300, total.lib.size = FALSE)
```

`expr` *Vector of expression differences*

Description

The t-statistics of differences in expression for genes on chromosome 21 between prostate cancer and normal epithelial cells.

Usage

```
expr.subset
```

Format

A numeric matrix, 309 rows and 1 column.

`FastQC-class` *FastQC and associated classes*

Description

The FastQC class stores results obtained from the FastQC application (see `references`), with a slot for each FastQC module. The SequenceQC class contains the QC results of a single lane of sequencing in three slots: `Unaligned` - FastQC results obtained from all reads (before alignment) `Aligned` - FastQC results obtained from only reads which aligned `Mismatches` - a `data.frame` containing counts for the number of mismatches of each type found at each sequencing cycle

Slots of a FastQC object

Basic_Statistics

Per_base_sequence_quality

Per_sequence_quality_scores

Per_base_sequence_content

Per_base_GC_content

Per_sequence_GC_content

Per_base_N_content

Sequence_Length_Distribution

Sequence_Duplication_Levels

Overrepresented_sequences

Slots of a SequenceQC object

Unaligned - FastQC results obtained from all reads (before alignment)

Aligned - FastQC results obtained from only reads which aligned

Mismatches - a data.frame containing counts for the number of mismatches of each type found at each sequencing cycle

MismatchTable - a data.frame containing counts of how many mismatches aligned sequences contain

Author(s)

Aaron Statham

References

FastQC - http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/

---

featureBlocks *Make windows for distances around a reference point.*

Description

Windows are made around a reference point, which is the start coordinate for features on the + strand, and the end coordinate for features on the - strand. For unstranded features, the reference point is taken to be the mid-point of the feature.

Usage

```
  ## S4 method for signature 'data.frame'
featureBlocks(anno, ...)
  ## S4 method for signature 'GRanges'
featureBlocks(anno, up = NULL, down = NULL, dist = c("base", "percent"),
                                    keep.strand = FALSE)
```

Arguments

anno	A data.frame or GRanges, describing some genomic features.
up	The amount to go upstream or towards the start of a chromosome. Semantics depend on the value of dist. See details.
down	The amount to go downstream or towards the end of a chromosome. Semantics depend on the value of dist. See details.
dist	Whether up and down refer to bases, or a percentage of each feature's width.
keep.strand	Whether the blocks should keep the strands of their features, or if all blocks should have strand be '*'
...	Arguments from the list above that are not used directly within the data.frame method.

Details

up refers to how many bases to go upstream for stranded features, or for unstranded features, how many bases to go towards the start of the chromosome, from the mid-point of the feature. Having a negative value for up means that the windows will start downstream by that amount, for stranded features. For unstranded features, it will start that many bases closer to the end of the chromosome, relative to the feature mid-point.

down is defined analogously.

Value

A GRanges of windows surrounding reference points for the features described by anno.

Author(s)

Dario Strbenac

Examples

```
genes <- data.frame(chr = c("chr1", "chr3", "chr7", "chr22"),
                    start = seq(1000, 4000, 1000),
                    end = seq(1500, 4500, 1000),
                    strand = c('+', '-', '-', '+'))

featureBlocks(genes, 500, 500)
```

`featureScores` *Get scores at regular sample points around genomic features.*

Description

Given a `GRanges` / `GRangesList` object, or BAM file paths, of reads for each experimental condition, or a `matrix` or an `AffynetrixCelSet`, or a numeric matrix of array data, where the rows are probes and the columns are the different samples,and an anntotation of features of interest, scores at regularly spaced positions around the features is calculated. In the case of sequencing data, it is the smoothed coverage of reads divided by the library size. In the case of array data, it is array intensity.

Usage

The ANY,data.frame method:

```
featureScores(x, anno, ...)
```

The ANY,GRanges method:

```
featureScores(x, anno, up = NULL, down = NULL, ...)
```

Arguments

x: Paths to BAM files, a collection of mapped short reads, or a collection of microarray data.

anno: Annotation of the features to sample around.

p.anno: A `data.frame` with columns `chr`, `position`, an optionally `index`. Only provide this if `x` is array data. If `index` is not provided, the rows are assumed to be in the same order as the elements of `x`.

mapping: A mapping between probes and genes, as made by `annotationLookup`. Avoids recomputing the mapping if it has already been done. Only provide this if `x` is array data.

chrs: A mapping between chromosome names in an ACP file to the user's feature annotation. Only provide this if `x` is an `AffymetrixCelSet`. There is no need to provide this if the feature annotation uses the same chromosome names as the ACP files do. Element `i` of this vector is the name to give to the chromosome numbered `i` in the ACP information.

up: How far to go from the features' reference points in one direction.

down: How far to go from the features' reference points in the opposite direction.

dist: The type of distance measure to use, in determining the boundaries of the sampling area. Only provide this if `x` is sequencing data. Default: `"base"`. `"percent"` is also accepted.

freq: Score sampling frequency.

log2.adj: Whether to log2 scale the array intensities. Only provide this if `x` is array data. Default: TRUE.

s.width: The width of smoothing to apply to the coverage. Only provide this if `x` is sequencing data. This argument is optional. If not provided, then no smoothing is done.

mappability: A `BSgenome` object, or list of such objects, the same length as `x` that has bases for which no mappable reads start at masked by N. If this was provided, then either `s.width` or `tag.len` must be provided (but not both).

map.cutoff: The percentage of bases in a window around each sampling position that must be mappable. Otherwise, the score at that position is repalced by NA. Default: 0.5

tag.len: Provide this if `mappability` was provided, but `s.width` was not.

use.strand: Whether to only consider reads on the same strand as the feature. Useful for RNA-seq applications.

verbose: Whether to print the progess of processing. Default: TRUE.

Details

If `x` is a vector of paths or `GRangesList` object, then `names(x)` should contain the types of the experiments.

If `anno` is a `data.frame`, it must contan the columns `chr`, `start`, and `end`. Optional columns are `strand` and `name`. If `anno` is a `GRanges` object, then the name can be present as a column called `name` in the element metadata of the GRanges object. If names are given, then the coverage matrices will use the names as their row names.

An approximation to running mean smoothing of the coverage is used. Reads are extended to the smoothing width, rather than to their fragment size, and coverage is used directly. This method is faster than a running mean of the calculated coverage, and qualtatively almost identical.

If providing a matrix of array intensity values, the column names of this matrix are used as the names of the samples.

The annotation can be stranded or not. if the annotation is stranded, then the reference point is the start coordinate for features on the + strand, and the end coordinate for features on the - strand. If the annotation is unstranded (e.g. annotation of CpG islands), then the midpoint of the feature is used for the reference point.

The `up` and `down` values give how far up and down from the reference point to find scores. The semantics of them depend on if the annotation is stranded or not. If the annotation is stranded, then they give how far upstream and downstream will be sampled. If the annotation is unstranded, then `up` gives how far towards the start of a chromosome to go, and `down` gives how far towards the end of a chromosome to go.

If sequencing data is being analysed, and `dist` is `"percent"`, then they give how many percent of each feature's width away from the reference point the sampling boundaries are. If `dist` is `"base"`, then the boundaries of the sampling region are a fixed width for every feature, and the units of `up` and `down` are bases. `up` and `down` must be identical if the features are unstranded. The units of `freq` are percent for `dist` being `"percent"`, and bases for `dist` being `"base"`.

In the case of array data, the sequence of positions described by `up`, `down`, and `freq` actually describe the boundaries of windows, and the probe that is closest to the midpoint of each window is chosen as the representative score of that window. On the other hand, when analysing sequencing data, the sequence of positions refer to the positions that coverage is taken for.

Providing a mappability object for sequencing data is recommended. Otherwise, it is not possible to know if a score of 0 is because the window around the sampling position is unmappable, or if there were really no reads mapping there in the experiment. Coverage is normalised by dividing the raw coverage by the total number of reads in a sample. The coverage at a sampling position is multiplied by 1 / mappability. Any positions that have mappabilty below the mappability cutoff will have their score set to NA.

Value

A ScoresList object, that holds a list of score matrices, one for each experiment type, and the parameters that were used to create the score matrices.

Author(s)

Dario Strbenac, with contributions from Matthew Young at WEHI.

See Also

mergeReplicates for merging sequencing data replicates of an experiment type.

Examples

```
data(chr21genes)
data(samplesList) # Loads 'samples.list.subset'.

fs <- featureScores(samples.list.subset[1:2], chr21genes, up = 2000, down = 1000,
                    freq = 500, s.width = 500)
```

---

findClusters *Find Clusters Epigenetically Modified Genes*

Description

Given a table of gene positions that has a score column, genes will first be sorted into positional order and consecutive windows of high or low scores will be reported.

Usage

```
findClusters(stats, score.col = NULL, w.size = NULL, n.med = NULL, n.consec = NULL,
          cut.samps = NULL, maxFDR = 0.05, trend = c("down", "up"), n.perm = 100,
              getFDRs = FALSE, verbose = TRUE)
```

Arguments

stats	A data.frame with (at least) column chr, and a column of scores. Genes must be sorted in positional order.
score.col	A number that gives the column in stats which contains the scores.
w.size	The number of consecutive genes to consider windows over. Must be odd.
n.med	Minimum number of genes in a window, that have median score centred around them above a cutoff.
n.consec	Minimum cluster size.
cut.samps	A vector of score cutoffs to calculate the FDR at.
maxFDR	The highest FDR level still deemed to be significant.

trend	Whether the clusters must have all positive scores (enrichment), or all negative scores (depletion).
n.perm	How many random tables to generate to use in the FDR calculations.
getFDRs	If TRUE, will also return the table of FDRs at a variety of score cutoffs, from which the score cutoff for calling clusters was chosen.
verbose	Whether to print progress of computations.

Details

First, the median over a window of size w.size is calculated in a rolling window and then associated with the middle gene of the window. Windows are again run over the genes, and the gene at the centre of the window is significant if there are also at least n.med genes with representative medians above the score cutoff, in the window that surrounds it. These marker genes are extended outwards, for as long as the score has the same sign. The order of the stats rows is randomised, and this process in done for every randomisation.

The procedure for calling clusters is done at a range of score cutoffs. The first score cutoff to give an FDR below maxFDR is chosen as the cutoff to use, and clusters are then called based on this cutoff.

Value

If getFDRs is FALSE, then only the stats table, with an additional column, cluster. If getFDRs is TRUE, then a list with elements :

table	The table stats with the additional column cluster.
FDR	The table of score cutoffs tried, and their FDRs.

Author(s)

Dario Strbenac, Aaron Statham

References

Saul Bert, in preparation

Examples

```
chrs <- sample(paste("chr", c(1:5), sep = ""), 500, replace = TRUE)
starts <- sample(1:10000000, 500, replace = TRUE)
ends <- starts + 10000
genes <- data.frame(chr = chrs, start = starts, end = ends, strand = '+')
genes <- genes[order(genes$chr, genes$start), ]
genes$t.stat = rnorm(500, 0, 2)
genes$t.stat[21:30] = rnorm(10, 4, 1)
findClusters(genes, 5, 5, 2, 3, seq(1, 10, 1), trend = "up", n.perm = 2)
```

GCadjustCopy *Calculate Absolute Copy Number from Sequencing Counts*

Description

Taking into account mappability and GC content biases, the absolute copy number is calculated, by assuming that the median read depth is a copy number of 1.

Usage

```
  ## S4 method for signature 'data.frame,matrix,GCAdjustParams'
GCadjustCopy(input.windows, input.counts,
                                                    gc.params, ...)
  ## S4 method for signature 'GRanges,matrix,GCAdjustParams'
GCadjustCopy(input.windows, input.counts,
                                                gc.params, verbose = TRUE)
```

Arguments

input.windows	A data.frame with (at least) columns chr, start, and end, or a GRanges object.
input.counts	A matrix of counts. Rows are genomic windows and columns are samples.
gc.params	A GCAdjustParams object, holding parameters related to mappability and GC content correction of read counts.
...	verbose argument, if data.frame method called.
verbose	Whether to print the progess of processing.

Details

First, the mappability of all counting windows is calculated, and windows that have mappability less than the cutoff specified by in the parameters object are ignored in further steps. The remaining windows have their counts scaled by multiplying their counts by 100 / percentage mappability.

The range of GC content of the counting windows is broken into a number of bins, as specified by the user in the parameters object. A probability density function is fitted to the counts in each bin, so the mode can be found. The mode is taken to be the counts of the copy neutral windows, for that GC content bin.

A polynomial function is fitted to the modes of GC content bins. Each count is divided by its expected counts from the polynomial function to give an absolute copy number estimate. If the ploidy has been provided in the parameters object, then all counts within a sample are multiplied by the ploidy for that sample. If the sample ploidys were omitted, then no scaling for ploidy is done.

Value

A AdjustedCopyEstimate object describing the input windows and their estimates.

Author(s)

Dario Strbenac

Examples

```
## Not run:
  library(BSgenome.Hsapiens.UCSC.hg18)
  library(BSgenome.Hsapiens36bp.UCSC.hg18mappability)
  load("inputsReads.RData")
  windows <- genomeBlocks(Hsapiens, chrs = paste("chr", c(1:22, 'X', 'Y'), sep = ''),
                          width = 20000)
  counts <- annotationBlocksCounts(inputsReads, anno = windows, seq.len = 300)

  gc.par <- GCAdjustParams(genome = Hsapiens, mappability = Hsapiens36bp,
                           min.mappability = 50, n.bins = 10, min.bin.size = 10,
                           poly.degree = 4, ploidy = c(2, 4))
 abs.cn <- GCadjustCopy(input.windows = windows, input.counts = counts, gc.params = gc.par)

## End(Not run)
```

GCAdjustParams *Container for parameters for mappability and GC content adjusted absolute copy number estimation.*

Description

The parameters are used by the absoluteCN function.

Constructor

GCAdjustParams(genome, mappability, min.mappability, n.bins = NULL, min.bin.size = 1, poly.degree = NULL, ploidy = 1) Creates a GCAdjustParams object.

genome A BSgenome object of the species that the experiment was done for.

mappability A BSgenome object, or the path to a FASTA file generated by GEM mappability containing the mappability of each base in the genome.

min.mappability A number between 0 and 100 that is a cutoff on window mappability.

n.bins The number of GC content bins to divide the windows into, before finding the mode of counts in each window.

min.bin.size GC bins with less than this many count windows inside them will be ignored.

poly.degree The degree of the polynomial to fit to the GC bins' count modes.

ploidy A vector of multipliers to use on the estimated absolute copy number of each sample, if the number of sets of chromosomes is known.

Author(s)

Dario Strbenac

GCbiasPlots *Plot GC content vs. Read Counts Before Normalising, and GC content vs. Copy Estimates After Normalising.*

Description

Two plots on the same plotting page are made for each sample. The top plot has estimates of copy number separated by GC content before any GC correction was applied. The bottom plot shows the copy number estimates after GC correction was applied.

Usage

```
  ## S4 method for signature 'AdjustedCopyEstimate'
GCbiasPlots(copy, y.max = NULL, pch = 19,
        cex = 0.2, pch.col = "black", line.col = "red", lty = 1, lwd = 2, verbose = TRUE)
```

Arguments

copy	A CopyEstimate object.
y.max	The maximum value of the y-axis of the scatter plots.
pch	Style of points in the scatter plots.
cex	Size of the points in the scatter plots.
pch.col	Colour of points in the scatter plots.
line.col	Colour of regression line in each scatter plot.
lty	Line type of plotted regression line.
lwd	Line width of plotted regression line.
verbose	Whether to print the progess of processing.

Details

See absoluteCN or relativeCN for how to do the GC adjusted copy number estimates. The line plotted through the scatterplots is a lowess line fit to the data points.

Value

A number of pages of scatterplots equal to the number of samples described by copy. The output should, therefore, be sent to a PDF device.

Author(s)

Dario Strbenac

Examples

```
## Not run:
  library(BSgenome.Hsapiens.UCSC.hg18)
  library(BSgenome.Hsapiens36bp.UCSC.hg18mappability)
  load("inputsReads.RData")
  windows <- genomeBlocks(Hsapiens, chrs = paste("chr", c(1:22, 'X', 'Y'), sep = ''),
                          width = 20000)
  counts <- annotationBlocksCounts(inputsReads, anno = windows, seq.len = 300)

  gc.par <- GCAdjustParams(genome = Hsapiens, mappability = Hsapiens36bp,
                           min.mappability = 50, n.bins = 10, min.bin.size = 10,
                           poly.degree = 4, ploidy = c(2, 4))
 abs.cn <- absoluteCN(input.windows = windows, input.counts = counts, gc.params = gc.par)

  pdf("bias.pdf")
  GCbiasPlots(abs.cn, y.max = 8)
  dev.off()

## End(Not run)
```

---

gcContentCalc *Calculate The gcContent of a Region*

Description

Function to calculate the GC content of windows

Usage

```
  ## S4 method for signature 'GRanges,BSgenome'
gcContentCalc(x, organism, verbose = TRUE)
  ## S4 method for signature 'data.frame,BSgenome'
gcContentCalc(x, organism, window = NULL, ...)
```

Arguments

x	A GRanges object or a data.frame, with columns chr and either position or start, end and strand.
window	Bases around the locations that are in the window. Calculation will consider windowSize/2 bases upstream, and windowSize / 2 - 1 bases downstream.
organism	The BSgenome object to calculate gcContent upon.
verbose	Whether to print the progess of processing.
...	The verbose variable for the data.frame method, passed onto the GRanges method.

Details

The windows considered will be `windowSize/2` bases upstream and `windowSize/2-1` bases downstream of the given position, for each position. The value returned for each region is a percentage of bases in that region that are a G or C.

Value

A vector of GC content percentages, one for each region.

Author(s)

Aaron Statham

Examples

```
require(BSgenome.Hsapiens.UCSC.hg18)
TSSTable <- data.frame(chr = paste("chr", c(1,2), sep = ""), position = c(100000, 200000))
gcContentCalc(TSSTable, 200, organism=Hsapiens)
```

`genomeBlocks` *Creates bins across a genome.*

Description

Creates a compact `GRanges` representation of bins across specified chromosomes of a given genome.

Usage

```
  ## S4 method for signature 'numeric'
genomeBlocks(genome, chrs = names(genome), width = NULL,
                                    spacing = width)
  ## S4 method for signature 'BSgenome'
genomeBlocks(genome, chrs = seqnames(genome), width = NULL,
                                     spacing = width)
```

Arguments

`genome`	Either a `BSgenome` object, or a named vector of integers (names being choromosome names, integers being the chromosome lengths), to get the chromosome lengths from.
`chrs`	A `vector` containing which chromosomes to create bins across. May either be numeric indicies or chromosome names. Default is all chromosomes given by `genome`.
`width`	The width in base pairs of each bin.
`spacing`	The space between the centres of each adjacent bin. By default, is equal to the `spacing` parameter, which gives non-overlapping bins. Values larger than `spacing` will give overlapping bins, and values smaller than `spacing` will give gaps between each bin.

Value

Returns a GRanges object, compatible with direct usage in annotationBlocksCounts

Author(s)

Aaron Statham

See Also

annotationBlocksCounts

Examples

```
chr.lengths <- c(800, 200, 200)
names(chr.lengths) <- c("chr1", "chr2", "chr3")
genomeBlocks(chr.lengths, width = 200)
```

---

genQC *Plot Quality Checking Information for Sequencing Data*

Description

A series of quality control plots for sequencing data are made.

Usage

```
  ## S4 method for signature 'character'
genQC(qc.data, ...)
  ## S4 method for signature 'SequenceQCSet'
genQC(qc.data, expt = "Experiment")
```

Arguments

qc.data	A vector of character strings, each containing an absolute path to an RData file of a SequenceQC object, or a SequenceQC set object.
expt	The names of the experiments which the lanes are about.
...	The expt argument, which is not directly used in the character method.

Details

qc.data can be named, in which case this gives the names of the lanes used in the plotting. Otherwise the lanes will be given the names "Lane 1", "Lane 2", ..., "Lane n".

Value

The function is called for its output. The output is multiple pages, so the pdf device should be called before this function is.

Author(s)

Dario Strbenac

References

FastQC: http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/

Examples

```
  ## Not run:
    qc.files <- list.files(qc.dir, "QC.*RData", full.names = TRUE)
    genQC(qc.files, "My Simple Experiment")

## End(Not run)
```

---

getProbePositionsDf *Translate Affymetrix probe information in a table.*

Description

Translates the probe information in the AromaCellPositionFile to a data.frame object.

Usage

```
  ## S4 method for signature 'AffymetrixCdfFile'
getProbePositionsDf(cdf, chrs, ..., verbose = TRUE)
```

Arguments

cdf	An AffymetrixCdfFile object.
chrs	A vector of chromosome names. Optional.
...	Further arguments to send to getCellIndices.
verbose	Logical; whether or not to print out progress statements to the screen.

Details

This assumes that the AromaCellPositionFile exist.

Value

A data.frame with 3 columns: chr, position, index

Author(s)

Mark Robinson

Examples

```
## not run
# probePositions <- getProbePositionsDf(cdfU)
```

`getSampleOffsets` *Calculates the sample-specific offsets, using the neutral state*

Description

ABCD-DNA combines CNV offsets with sample specific factors. This function calculates the latter, using a set of neutral regions (and corresponding counts in the count table).

Usage

```
getSampleOffsets(obj, ref = 1, quantile = 0.99, min.n = 100, plot.it = FALSE, force = FALSE, ...)
```

Arguments

`obj`	a `QdnaData` object
`ref`	integer index, giving the sample to use as reference
`quantile`	quantile of the A-values to use
`min.n`	minimum number of points to include
`plot.it`	logical, whether to plot an M-A plot for each sample against the reference (default: `FALSE`)
`force`	logical, whether to recalculate the sample-specific offsets (only needed if they are already calculated)
`...`	arguments to pass to the `maPlot` function

Details

The sample-specific offset is calculated as the median M-value beyond (i.e. to the right) an A-value quantile, using only the copy-number-neutral regions, as specified in the incoming `QdnaData` object.

Value

returns a `QdnaData` object (copied from the `obj` argument) and populates the `$DGEList$samples$norm.factors` element and sets the `$sample.specific.calculated` to `TRUE`.

Author(s)

Mark Robinson

References

http://imlspenticton.uzh.ch/robinson_lab/ABCD-DNA/ABCD-DNA.html

See Also

QdnaData

Examples

```
# library(Repitools)
# qd <- QdnaData(counts=counts, regions=gb, design=design,
#               cnv.offsets=cn, neutral=(regs=="L=4 P=2"))
# qd <- getSampleOffsets(qd,ref=1)
```

---

hcRegions *Masking files for hg19*

Description

File to mask out areas of the genome that are prone to causing false positives in ChIP-seq and other sequencing based functional assays, as proposed by Pickrell et al. (2011), Bioinformatics 27: 2144-2146, http://eqtl.uchicago.edu/Home.html.

Usage

```
hcRegions
```

Format

A GRanges object created using the bedfile provided on http://eqtl.uchicago.edu/Masking/seq.cov1.ONHG19.bed.gz.

Source

Pickrell et al. (2011), Bioinformatics 27: 2144-2146.

---

hyperg2F1_vec *Gaussian hypergeometric function for vectorial arguments*

Description

Computes the value of the Gaussian hypergeometric function 2_F_1 as defined in Abramowitz and Stegun (1972, page 558), i.e. for $|z| < 1$ and $c > b > 0$ using the Cephes library.

Usage

```
hyperg2F1_vec(a,b,c,z)
```

Arguments

a	(Vectorial) parameter a.
b	parameter b (of same length as a)
c	parameter c (of same length as a)
z	parameter z (of same length as a)

Details

The function is in particular efficient for vectorial arguments as the loop is shifted to C. Note: If vectorial arguments are provided, all arguments need to be of the same length.

Value

The value of the Gaussian hypergeometric function F(a,b,c,z) for c > b > 0 and |z| < 1.

Author(s)

Andrea Riebler and Daniel Sabanes Bove

References

Abramowitz and Stegun 1972. _Handbook of mathematical functions with formulas, graphs and mathematical tables_. New York: Dowver Publications.

www.netlib.org/cephes/

See Also

package hypergeo or BMS.

Examples

```
hyperg2F1_vec(-10.34,2.05,3.05,0.1725)
hyperg2F1_vec(30,1,20,.8) # returns about 165.8197
hyperg2F1_vec(30,10,20,0) # returns one
hyperg2F1_vec(10,15,20,-0.1) # returns about 0.4872972

hyperg2F1_vec(c(-10.34, 30, 10), c(2.05, 1, 10), c(3.05, 20, 20),
    c(0.1725, 0.8, 0))
hyperg2F1_vec(a=1.2+1:10/10, b=rep(1.4,10), c=rep(1.665,10), z=rep(.3,10))
```

`loadPairFile` *A routine to read Nimblegen tiling array intensities*

Description

Reads a file in Nimblegen pair format, returning log2 intensities of probes referenced by the supplied ndf data frame.

Usage

```
loadPairFile(filename = NULL, ndf = NULL, ncols = 768)
```

Arguments

`filename`	the name of the pair file which intensities are to be read from.
`ndf`	a data frame produced by `processNDF`.
`ncols`	the number of columns of probes on the array - must be the same value as used in `processNDF`. The default works for 385K format arrays.

Details

Reads in intensities from the specified pair file, then matches probes against those specified in the supplied ndf.

Value

a `vector` of log2 intensities, the number of rows of the supplied `ndf` in length.

Author(s)

Aaron Statham

See Also

`loadSampleDirectory` for reading multiple pair files with the same ndf. `processNDF`

Examples

```
# Not run
#
## Read in the NDF file
# ndfAll <- processNDF("080310_HG18_chr7RSFS_AS_ChIP.ndf")
#
## Subset the NDF to only probes against chromosomes
# ndf <- ndfAll[grep("^chr", ndfAll$chr),]
#
## Read in a pair file using the chromosome only NDF
# arrayIntensity <- loadPairFile("Pairs/Array1_532.pair", ndf)
#
```

`loadSampleDirectory` *A routine to read Nimblegen tiling array intensities*

Description

Reads all files in Nimblegen pair format within the specified directory, returning log2 intensities of probes referenced by the supplied ndf data frame.

Usage

```
loadSampleDirectory(path = NULL, ndf = NULL, what="Cy3", ncols = 768)
```

Arguments

`path`	the directory containing the pair files to be read.
`ndf`	a data frame produced by `processNDF`.
`what`	specifies the channel(s) to be read in - either `Cy3`, `Cy5`, `Cy3/Cy5`, `Cy5/Cy3`, `Cy3andCy5`, `Cy5andCy3`.
`ncols`	the number of columns of probes on the array - must be the same value as used in `processNDF`. The default works for 385K format arrays.

Details

Reads in intensities of all arrays contained within `path`. The parameter `what` determines which fluorescent channels are read, and how the are returned. `Cy3` and `Cy5` return the log2 intensity of the specified single channel. `Cy3/Cy5` and `Cy5/Cy3` return the log2 ratio of the two channels. `Cy3andCy5` and `Cy5andCy3` return the log2 intensity of both channels in separate columns of the matrix.

Value

a `matrix` of log2 intensites, with the same number of rows as the supplied `ndf` and depending on the value of `what` either one or two columns per array.

Author(s)

Aaron Statham

See Also

`loadPairFile` for reading a single pair files. `processNDF`

Examples

```
# Not run
#
## Read in the NDF file
# ndfAll <- processNDF("080310_HG18_chr7RSFS_AS_ChIP.ndf")
#
## Subset the NDF to only probes against chromosomes
# ndf <- ndfAll[grep("^chr", ndfAll$chr),]
#
## Read in a directory of pair files, returning both the Cy3 and Cy5 fluorescence in separate columns
# arrayIntensities <- loadSampleDirectory("Arrays", ndf, what="Cy3andCy5")
#
```

---

makeWindowLookupTable *Using the output of 'annotationLookup', create a tabular storage of the indices*

Description

To allow easy access to the probe-level data for either a gene, or an area of the promoter (over all genes), this routine takes the output of annotationLookup and organizes the indices into a table, one row for each gene and one column for each region of the promoter.

Usage

```
makeWindowLookupTable(indexes = NULL, offsets = NULL, starts = NULL, ends = NULL)
```

Arguments

indexes	a list of indices, e.g. indexes element from annotationLookup output
offsets	a list of offsets, e.g. offsets element from annotationLookup output
starts	a vector of starts
ends	a vector of ends

Details

The vectors starts and ends (which should be the same length) determine the number of columns in the output matrix.

Value

A matrix with rows for each gene and columns for each bin of the promoter. NA signifies that there is no probe in the given distance from a TSS.

Author(s)

Mark Robinson

See Also

annotationLookup

Examples

```
# create example set of probes and gene start sites
probeTab <- data.frame(position=seq(1000,3000,by=200), chr="chrX", strand = '-')
genes <- data.frame(chr="chrX", start=c(2100, 1000), end = c(3000, 2200), strand=c("+","-"))
rownames(genes) <- paste("gene",1:2,sep="")

# Call annotationLookup() and look at output
aL <- annotationLookup(probeTab, genes, 500, 500)
print(aL)

# Store the results of annotationLookup() in a convenient tabular format
lookupTab <- makeWindowLookupTable(aL$indexes, aL$offsets, starts=seq(-400,200,by=200), ends=seq(-200,400,by=200
print(lookupTab)
```

---

mappabilityCalc *Calculate The Mappability of a Region*

Description

Function to calculate mappability of windows

Usage

```
  ## S4 method for signature 'GRanges,MappabilitySource'
mappabilityCalc(x, organism, window = NULL,
              type = c("block", "TSS", "center"), verbose = TRUE)
  ## S4 method for signature 'data.frame,MappabilitySource'
mappabilityCalc(x, organism, window = NULL,
                                      type = c("block", "TSS", "center"), ...)
```

Arguments

x	A GRanges object or a data.frame, with columns chr and either position or start, end and strand.
window	Bases around the locations that are in the window. Calculation will consider windowSize/2 bases upstream, and windowSize/2-1 bases downstream.

For unstranded features, the effect is the same as for + strand features.

type	What part of the interval to make the window around. If the value is "TSS", the the start coordinate is used for all + strand features, and the end coordinate is used for all - strand features. If "cemter" is chosen, then the coordinate that is half way between the start and end of each feature will be used as the reference point. "block" results in the use the start and end coordinates without modification.

organism	The BSgenome object to calculate mappability upon, or the file path to a FASTA file generated by GEM Mappability, or the path to a bigWig file containing mappability scores.
verbose	Whether to print the progess of processing.
...	The verbose variable for the data.frame method, passed onto the GRanges method.

Details

The windows considered will be windowSize/2 bases upstream and windowSize/2-1 bases downstream of the given position of stranded features, and the same number of bases towards the start and end of the chromosome for unstranded features. The value returned for each region is a percentage of bases in that region that are not N (any base in IUPAC nomenclature).

For any positions of a window that are off the end of a chromosome, they will be considered as being N.

Value

A vector of mappability percentages, one for each region.

Author(s)

Aaron Statham

Examples

```
## Not run:
  require(BSgenome.Hsapiens36bp.UCSC.hg18mappability)
 TSSTable <- data.frame(chr = paste("chr", c(1,2), sep = ""), position = c(100000, 200000))
  mappabilityCalc(TSSTable, Hsapiens36bp, window = 200, type = "TSS")

## End(Not run)
```

MappabilitySource *Superclass for datatypes that can refer to genome mappability data.*

Description

This class is simply the union of character and BSgenome classes.

Author(s)

Dario Strbenac

maskOut *Function to mask suspicious regions.*

Description

Function to mask out regions that are prone to causing problems in the empirical Bayes approach empBayes. The corresponding bins are marked and in the empirical Bayes approach not taken into account. Notice that methylation estimates using methylEst will nevertheless be produced for these bins.

Usage

```
maskOut(x, ranges)
```

Arguments

x	Object of class BayMethList.
ranges	A GRanges object definining the coordinates of regions to be masked out.

Value

A BayMethList object where the slot maskout is filled with a boolean vector indicating which bins will be excluded in empBayes.

Author(s)

Andrea Riebler

Examples

```
if(require(BSgenome.Hsapiens.UCSC.hg18)){
    windows <- genomeBlocks(Hsapiens, chrs="chr21", width=100, spacing=100)
    cpgdens <- cpgDensityCalc(windows, organism=Hsapiens,
        w.function="linear", window=700)
    co <- matrix(rnbinom(length(windows), mu=10, size=2), ncol=1)
    sI <- matrix(rnbinom(2*length(windows), mu=5, size=2), ncol=2)
    bm <- BayMethList(windows=windows, control=co,
        sampleInterest=sI, cpgDens=cpgdens)

    # mask out unannotated high copy number regions
    # see Pickrell et al. (2011), Bioinformatics 27: 2144-2146.
    data(hcRegions)

    bm <- maskOut(bm, hcRegions)
}
```

mergeReplicates *Merge GRanges that are of replicate experiments.*

Description

A lane of next generation sequencing data can be stored as a GRanges object. Sometimes, a GRangesList of various lanes can have experimental replicates. This function allows the merging of such elements.

Usage

```
  ## S4 method for signature 'GRangesList'
mergeReplicates(reads, types, verbose = TRUE)
```

Arguments

reads	A GRangesList.
types	A vector the same length as reads, that gives what type of experiment each element is of.
verbose	Whether to print the progess of processing.

Details

The experiment type that each element of the merged list is of, is stored in the first element of the metadata list.

Value

A GRangesList with one element per experiment type.

Author(s)

Dario Strbenac

Examples

```
library(GenomicRanges)
grl <- GRangesList(GRanges("chr1", IRanges(5, 10)),
                   GRanges("chr18", IRanges(25, 50)),
                   GRanges("chr22", IRanges(1, 100)))
antibody <- c("MeDIP", "MeDIP", "H3K4me3")
mergeReplicates(grl, antibody)
```

`methylEst` *Function to derive regional methylation estimates.*

Description

Posterior mean and variance for the regional methylation level are derived for all genomic regions. Credible intervals can be computed either numerically from the posterior marginal distribution or by computing them on logit scale and transferring them back.

Usage

```
methylEst(x, verbose=FALSE, controlCI = list(compute = FALSE, method = "Wald",
    level = 0.95, nmarg = 512, ncpu = NULL))
```

Arguments

`x`	Object of class `BayMethList`.
`verbose`	Boolean indicating whether the methylEst function should run in a verbose mode (default 'FALSE').
`controlCI`	list defining whether credible intervals should be derived.

- `compute` logical. If 'TRUE' credible intervals are derived. (default FALSE)
- `method` There are three possible types of credible intervals that can be chosen if a uniform prior, e.g. Beta(1,1), is chosen: 'Wald' (default), 'HPD', 'quantile'. The Wald-type intervals are the fastest to compute. The are calculated on logit scale and then transferred back. Due to numerical integration of the posterior marginal posterior distributions, the computation of highest posterior density (HPD) interval and quantile-based interval is computationally more expensive. However, in our applications HPD intervals provided best coverage.
  Note, using a beta mixture or a Dirac-beta-Dirac (DBD) mixture as prior distribution for the methylation level only `method="quantile"` is available.
- `level` numerical value defining the credible level. Default: 0.95.
- `nmarg` Number of points at which the posterior marginal is evaluated (only relevant for method="quantile" or method="HPD").
- `ncpu` Number of CPUs on your machine you would like to use in parallel. If `ncpu` is set to NULL, half of the CPUs will be used on machines with a maximum of four CPUs, and 2/3 will be used if more are available.

Details

The posterior mean and the variance are analytically available and therefore straightforward to efficiently compute; Wald-based credible intervals are obtained on logit scale and then back-transferred to ensure values withing 0 and 1. HPD and quantile-based credible intervals are computed by numerical integration of the posterior marginal distribution.

Value

A BayMethList object where the slot methEst is filled with a list containing the following elements:

mean	Matrix where the number of columns equals the number of samples of interest. Each column contains the posterior mean methylation level for each bin.
var	Matrix where the number of columns equals the number of samples of interest. Each column contains posterior variance for each bin.
ci	List with length equal to the number of samples of interest. Each list element contains a matrix where the first column contains the lower CI bound and the second column the upper CI bound.
W	Matrix where the number of columns equals the number of samples of interest. Each column contains the normalisation factor of the posterior marginal distribution for each bin.
al	Matrix where the number of columns equals the number of samples of interest. Each column contains the prior shape parameter for each bin
bl	Matrix where the number of columns equals the number of samples of interest. Each column contains the prior scale parameter for each bin

Author(s)

Andrea Riebler

Examples

```
if(require(BSgenome.Hsapiens.UCSC.hg18)){
    windows <- genomeBlocks(Hsapiens, chrs="chr21", width=100, spacing=100)
    cpgdens <- cpgDensityCalc(windows, organism=Hsapiens,
        w.function="linear", window=700)
    co <- matrix(rnbinom(length(windows), mu=10, size=2), ncol=1)
    sI <- matrix(rnbinom(2*length(windows), mu=5, size=2), ncol=2)
    bm <- BayMethList(windows=windows, control=co,
        sampleInterest=sI, cpgDens=cpgdens)

    bm <- determineOffset(bm)
    # should take about 3 minutes for both samples of interests with 2 CPUs.
    bm <- empBayes(bm)
    bm <- methylEst(bm, controlCI = list(compute = FALSE, method = "Wald",
        level = 0.95, nmarg = 512, ncpu = NULL))
}
```

---

multiHeatmap *Superfigure plots*

Description

This function takes a list of matrices and plots heatmaps for each one. There are several features for the spacing (X and Y), colour scales, titles and label sizes. If a matrix has row and/or column names, these are added to the plot.

Usage

```
multiHeatmap(dataList, colourList, titles = NULL, main = "", showColour = TRUE, xspace = 1, cwidth = 0.5,
```

Arguments

dataList	A list of matrices to be plotted as different panels
colourList	A list of colourscales (if length 1, it is copied for all panels of the plot)
titles	A vector of panel titles
main	A main title
showColour	logical or logical vector, whether to plot the colour scale
xspace	The space between the panels (relative to number of columns). This can be either a scalar or a vector of length(dataList)+1
cwidth	widths of the colour scales relative to the width of the panels
ystarts	A vector of length 5 of numbers between 0 and 1 giving the relative Y positions of where the heatmaps, colourscale labels, colour scales, panel titles and main title (respectively) start
rlabelcex	character expansion factor for row labels
clabelcex	character expansion factor for column labels
titlecex	character expansion factor for panel titles
maincex	character expansion factor for main title
scalecex	character expansion factor for colour scale labels
offset	small offset to adjust scales for point beyond the colour scale boundaries

Value

This function is called for its output, a plot in the current device.

Author(s)

Mark Robinson

Examples

```
library(gplots)

cL <- NULL
br <- seq(-3,3,length=101)
col <- colorpanel(low="blue",mid="grey",high="red",n=101)
cL[[1]] <- list(breaks=br,colors=col)
br <- seq(-2,2,length=101)
col <- colorpanel(low="green",mid="black",high="red",n=101)
cL[[2]] <- list(breaks=br,colors=col)
br <- seq(0,20,length=101)
col <- colorpanel(low="black",mid="grey",high="white",n=101)
cL[[3]] <- list(breaks=br,colors=col)
```

```
testD <- list(matrix(runif(400),nrow=20),matrix(rnorm(100),nrow=20),matrix(rpois(100,lambda=10),nrow=20))
colnames(testD[[1]]) <- letters[1:20]
rownames(testD[[1]]) <- paste("row",1:20,sep="")

multiHeatmap(testD,cL,xspace=1)
```

`plotClusters` *Plot Scores of Cluster Regions*

Description

Given an annotation of gene positions that has a score column, the function will make a series of bar chart plots, one for each cluster.

Usage

```
  ## S4 method for signature 'data.frame'
plotClusters(x, s.col = NULL, non.cl = NULL, ...)
  ## S4 method for signature 'GRanges'
plotClusters(x, s.col = NULL, non.cl = NULL, ...)
```

Arguments

`x`	A summary of genes and their statistical score, and the cluster that they belong to. Either a `data.frame` or a `GRanges`. If a `data.frame`, then (at least) columns `chr`, `start`, `end`, `strand`, `name` and `cluster`. Also a score column, with the column name describing what type of score it is. If a `GRanges`, then the `elementMetadata` should have a `DataFrame` with a score column, and columns named `"cluster"` and `"name"`.
`s.col`	The column number of the `data.frame` when `data` is a `data.frame`, or the column number of the `DataFrame` when `data` is a `GRanges` object. The name of this column is used as the y-axis label in the plot.
`non.cl`	The value in the cluster column that represents genes that are not in any cluster
`...`	Further parameters to be passed onto `plot`.

Value

A plot for each cluster is made. Therefore, the PDF device should be opened before this function is called.

Author(s)

Dario Strbenac

Examples

```
library(GenomicRanges)
g.summary <- GRanges("chr1",
                     IRanges(seq(1000, 10000, 1000), width = 100),
                     rep(c('+', '-'), 5),
                     `t-statistic` = rnorm(10, 8, 2),
                     cluster = c(0, 0, 0, 0, 0, 1, 1, 1, 1, 0),
                     name = paste("Gene", 1:10))
plotClusters(g.summary, 1, 0, ylim = c(4, 12), lwd = 5)
```

`plotQdnaByCN` *Plotting the response of qDNA-seq data by CNV*

Description

Given groupings of relative CNV state, this function produces M-A (log-fold-change versus log-average) plots to compare two samples relative read densities. In addition, it calculates a scaling factor at a specified quantile and plots the median M value across all the groups.

Usage

```
plotQdnaByCN(obj, cnv.group, idx.ref = 1, idx.sam = 2, min.n = 100, quantile = 0.99, ylim = c(-5, 5), ...
```

Arguments

`obj`	a `QdnaData` object
`cnv.group`	a `character` vector or `factor` giving the relative CNV state. This must be the same length as the number of regions in `obj`
`idx.ref`	index of the reference sample (denominator in the calculation of M values)
`idx.sam`	index of the sample of interest (numerator in the calculation of M values)
`min.n`	minimum number of points to include
`quantile`	quantile of the A-values to use
`ylim`	y-axis limits to impose on all M-A plots
`...`	further arguments sent to `maPlot`

Value

a plot to the current graphics device

Author(s)

Mark Robinson

References

http://imlspenticton.uzh.ch/robinson_lab/ABCD-DNA/ABCD-DNA.html

See Also

QdnaData, ~~~

Examples

```
# library(Repitools)
# qd <- QdnaData(counts=counts, regions=gb, design=design,
#                cnv.offsets=cn, neutral=(regs=="L=4 P=2"))
# plotQdnaByCN(qd,cnv.group=regs,idx.ref=3,idx.sam=2)
```

---

processNDF *Reads in a Nimblegen microarray design file (NDF)*

Description

Reads a Nimblegen microarray design file (NDF file) which describes positions and sequences of probes on a Nimblegen microarray.

Usage

```
processNDF(filename = NULL, ncols = 768)
```

Arguments

filename	the name of the Nimblegen microarray design file
ncols	the number of columns of probes on the array - must be the same value as will be passed to loadPairFile or loadSampleDirectory. The default works for 385K format arrays.

Details

Reads in a Nimblegen microarray design file. This enables the reading in and annotation of Nimblegen microarray data files (pair files).

Value

a data frame containing

chr	the chromosome the probe was designed against
position	the position of the sequence the probe was designed against (probe centre)
strand	the strand the probe was designed against
index	the index (x y position) the probe occupies on the array
sequence	the actual DNA sequence synthesised onto the array
GC	the percent GC content of the probe sequence

Author(s)

Aaron Statham

See Also

loadSampleDirectory, loadPairFile

Examples

```
# Not run
#
## Read in the NDF file
# ndfAll <- processNDF("080310_HG18_chr7RSFS_AS_ChIP.ndf")
#
## Subset the NDF to only probes against chromosomes
# ndf <- ndfAll[grep("^chr", ndfAll$chr),]
```

---

profilePlots — *Create line plots of averaged signal across a promoter for gene sets, compared to random sampling.*

Description

Creates a plot where the average signal across a promoter of supplied gene lists is compared to random samplings of all genes, with a shaded confidence area.

Usage

```
  ## S4 method for signature 'ScoresList'
profilePlots(x, summarize = c("mean", "median"), gene.lists,
  n.samples = 1000, confidence = 0.975, legend.plot = "topleft", cols = rainbow(length(gene.lists)),
   verbose = TRUE, ...)
```

Arguments

x	A ScoresList object. See featureScores.
summarize	How to summarise the scores for each bin into a single value.
gene.lists	Named list of logical or integer vectors, specifying the genes to be averaged and plotted. NAs are allowed if the vector is logical.
n.samples	The number of times to randomly sample from all genes.
confidence	A percentage confidence interval to be plotted (must be > 0.5 and < 1.0).
legend.plot	Where to plot the legend - directly passed to legend. NA suppresses the legend.
cols	The colour for each of the genelists supplied.
verbose	Whether to print details of processing.
...	Extra arguments to matplot, like x- and y-limits, perhaps.

Details

For each table of scores in `x`, a plot is created showing the average signal of the genes specified in each list element of `gene.lists` compared to `n.samples` random samplings of all genes, with `confidence` % intervals shaded. If an element of `gene.lists` is a `logical` vector, its length must be the same as the number of rows of the score tables.

Value

A series of plots.

Author(s)

Aaron Statham

Examples

```
# See examples in manual.
```

`QdnaData` *A container for quantitative DNA sequencing data for ABCD-DNA analyses*

Description

`QdnaData` objects form the basis for differential analyses of quantitative DNA sequencing data(i.e. ABCD-DNA). A user is required to specify the minimum elements: a count table, a list of regions and a design matrix. For copy-number-aware analyses, a table of offsets and the set of neutral regions needs to be given.

Usage

```
QdnaData(counts, regions, design, cnv.offsets = NULL, neutral = NULL)
```

Arguments

`counts`	table of counts for regions of interest across all samples
`regions`	a `GRanges` object giving the regions
`design`	a design matrix
`cnv.offsets`	a table of offsets. If unspecified (or `NULL`), a matrix of 1s (i.e. no CNV) is used
`neutral`	a logical vector, or indices, of the regions deemed to be neutral. If unspecified (or `NULL`), all regions are used

Details

`QdnaData` objects are geared for general differential analyses of qDNA-seq data. If CNV is present and prominent, the objects and methods available with `QdnaData` perform adjustments and spot checks before the differential analysis.

Value

a QdnaData object (effectively a list) is returned

Author(s)

Mark Robinson

References

http://imlspenticton.uzh.ch/robinson_lab/ABCD-DNA/ABCD-DNA.html

See Also

getSampleOffsets, plotQdnaByCN, setCNVOffsets

Examples

```
require(GenomicRanges)
cnt <- matrix(rpois(20,lambda=10),ncol=4)
gr <- GRanges("chr1",IRanges(seq(2e3,6e3,by=1e3), width=500))
des <- model.matrix(~c(0,0,1,1))
qd <- QdnaData( counts=cnt, regions=gr, design=des)
```

---

regionStats *Find Regions of significance in microarray data*

Description

The function finds the highest smoothed score cutoff for a pre-specified FDR. Smoothing is performed over a specified number of basepairs, and regions must have a minimum number of qualifying probes to be considered significant. The FDR is calculated as the ratio of the number of significant regions found in a permutation-based test, to the number found in the actual experimental microarray data.

Usage

```
  ## S4 method for signature 'matrix'
regionStats(x, design = NULL, maxFDR=0.05, n.perm=5, window=600, mean.trim=.1, min.probes=10, max.gap=
  ## S4 method for signature 'AffymetrixCelSet'
regionStats(x, design = NULL, maxFDR=0.05, n.perm=5, window=600, mean.trim=.1, min.probes=10, max.gap=
```

Arguments

x	An AffymetrixCelSet or matrix of array data to use.
design	A design matrix of how to manipulate
maxFDR	Cutoff of the maximum acceptable FDR
n.perm	Number of permutations to use

window	Size of window, in base pairs, to check for
mean.trim	A number representing the top and bottom fraction of ordered values in a window to be removed, before the window mean is calculated.
min.probes	Minimum number of probes in a window, for the region to qualify as a region of significance.
max.gap	Maximum gap between significant probes allowable.
two.sides	Look for both significant positive and negative regions.
ind	A vector of the positions of the probes on the array
ndf	The Nimblegen Definition File for Nimblegen array data.
return.tm	If TRUE, the values of the trimmed means of the intensities and permuted intensities are also retuned from the function.
verbose	Whether to print the progress of processing.

Value

A RegionStats object (list) with elements

regions	A list of data.frame. Each data.frame has columns chr, start, end, score.
tMeanReal	Matrix of smoothed scores of intensity data. Each column is an experimental design.
tMeanPerms	Matrix of smoothed scores of permuted intensity data. Each column is an experimental design.
fdrTables	List of table of FDR at different score cutoffs. Each list element is for a different experimental design.

Author(s)

Mark Robinson

Examples

```
## Not run:
library(Repitools)
library(aroma.affymetrix)

# assumes appropriate files are at annotationData/chipTypes/Hs_PromPR_v02/
cdf <- AffymetrixCdfFile$byChipType("Hs_PromPR_v02",verbose=-20)
cdfU <- getUniqueCdf(cdf,verbose=-20)

# assumes appropriate files are at rawData/experiment/Hs_PromPR_v02/
cs <- AffymetrixCelSet$byName("experiment",cdf=cdf,verbose=-20)
mn <- MatNormalization(cs)
csMN <- process(mn,verbose=-50)
csMNU <- convertToUnique(csMN,verbose=-20)

#> getNames(cs)
# [1] "samp1"  "samp2"  "samp3"  "samp4"
```

```
design <- matrix( c(1,-1,rep(0,length(cs)-2)), ncol=1, dimnames=list(getNames(cs),"elut5_L-P") )

# just get indices of chr7 here
ind <- getCellIndices(cdfU, unit = indexOf(cdfU, "chr7F"), unlist = TRUE, useNames = FALSE)

regs <- regionStats(csMNU, design, ind = ind, window = 500, verbose = TRUE)

## End(Not run)
```

`relativeCN` *Calculate and Segment Relative Copy Number From Sequencing Counts*

Description

This function uses the `GCadjustCopy` function to convert a matrix of count data into absolute copy number estimates, then calculates the log2 fold change ratio and segments these values.

Usage

```
  ## S4 method for signature 'data.frame,matrix'
relativeCN(input.windows, input.counts, gc.params = NULL,
                                        ..., verbose = TRUE)
  ## S4 method for signature 'GRanges,matrix'
relativeCN(input.windows, input.counts, gc.params = NULL,
                                       ..., verbose = TRUE)
```

Arguments

`input.windows`	A `data.frame` with (at least) columns `chr`, `start`, and `end`, or a GRanges object.
`input.counts`	A matrix of counts. The first column must be for the control state, and the second column must be for the treatment state.
`gc.params`	A `GCAdjustParams` object, holding parameters related to mappability and GC content correction of read counts, or NULL, if GC content correction is not desired.
`...`	Further parameters passed to `segment` function in `DNAcopy` package, and also the `segment.sqrt` parameter to `absoluteCN`.
`verbose`	Whether to print the progess of processing.

Details

The algorithm used to call the copy number regions is Circular Binary Segmentation (Olshen et al. 2004). Weights for each window, that are the inverse of the variance, calculated with the delta method, are always used. Windows or regions that were not in the segmentation result are given the value `NA`.

If gc.params is NULL, then no correction for mappability or GC content is done. This can be done when the bias in both treatment and control samples is assumed to be equal. If gc.params is specified, then absolute copy numbers are estimated with GCadjustCopy for each condition, which corrects for mappability and then GC content, before estimating absolute copy numbers. The ratio of estimated absolute copy numbers is segmented, to calculate relative copy numbers.

Value

If gc.params was given, then a AdjustedCopyEstimate object. Otherwise, a CopyEstimate object. The copy number ratios are on the linear scale, not log2.

Author(s)

Dario Strbenac

References

Olshen, A. B., Venkatraman, E. S., Lucito, R., and Wigler, M. (2004). Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics* 5: 557-572

Examples

```
inputs <- data.frame(chr = c("chr1", "chr1", "chr1", "chr2", "chr2"),
                     start = c(1, 50001, 100001, 1, 10001),
                       end = c(50000, 100000, 150000, 10000, 20000))
counts <- matrix(c(25, 39, 3, 10, 22, 29, 38, 5, 19, 31), nrow = 5)
colnames(counts) <- c("Control", "Treatment")
relativeCN(inputs, input.counts = counts, p.method = "perm")
```

---

samplesList *Short Reads from Cancer and Normal*

Description

Short reads that mapped to chromosome 21 in an Illumina sequencing experiment that was looking for differences between healthy epithelial and prostate cancer cells. The DNA was immunoprecipitated by a DNA methylation binding antibody.

Usage

```
samples.list.subset
```

Format

A GRangesList.

ScoresList *Container for* featureScores() *output.*

Description

Contains a list of tables of sequencing coverages or array intensities, and the parameters that were used to generate them.

Accessors

In the following code snippets, x is a ScoresList object.

names(x), names(x) <- value Gets and sets the experiment type names.

tables(x) Gets the list of score matrices.

length(x) Gets the number of score matrices.

Subsetting

In the following code snippets, x is a ScoresList object.

x[i] Creates a ScoresList object, keeping only the i matrices.

subsetRows(x, i = NULL) Creates a ScoresList object, keeping only the i features.

Author(s)

Dario Strbenac

sequenceCalc *Find occurences of a DNA pattern*

Description

Function to find all occurrences of a DNA pattern in given locations.

Usage

```
  ## S4 method for signature 'GRanges,BSgenome'
sequenceCalc(x, organism, pattern, fixed = TRUE, positions = FALSE)
  ## S4 method for signature 'data.frame,BSgenome'
sequenceCalc(x, organism, window = NULL, positions = FALSE, ...)
```

Arguments

x	A data.frame, with columns chr and position, or instead of the column position there can be columns start, end, and strand, or a GRanges object of the regions.
window	Bases around the locations supplied in x that are in the window. Calculation will consider windowSize/2-1 bases upstream, and windowSize/2 bases downstream.
organism	The BSgenome object to calculate CpG density upon.
pattern	The DNAString to search for.
fixed	Whether to allow degenerate matches.
positions	If TRUE FALSE
...	Arguments passed into the GRanges method

Details

If the version of the data frame with the start, end, and strand columns is given, the window will be created around the TSS.

Value

If positions is TRUE, a list of vectors of positions of matches in relation to the elements of x, otherwise a vector of the number of matches for each element of x.

Author(s)

Aaron Statham

See Also

cpgDensityCalc, mappabilityCalc, gcContentCalc

Examples

```
require(BSgenome.Hsapiens.UCSC.hg18)
TSSTable <- data.frame(chr=paste("chr",c(1,2),sep=""), position=c(100000,200000))
sequenceCalc(TSSTable, 600, organism=Hsapiens, pattern=DNAString("CG"))
```

---

setCNVOffsets *Set the CNVOffsets of a* QdnaData *object*

Description

A utility function to manually add CNV offset to a QdnaData object

Usage

```
setCNVOffsets(obj, cnv.offsets)
```

Arguments

obj	a QdnaData object
cnv.offsets	a matrix of offsets (presumably copy number)

Value

a QdnaData object

Author(s)

Mark Robinson

See Also

QdnaData

Examples

```
# library(Repitools)
# qd <- QdnaData(counts=counts, regions=gb, design=design,
#                neutral=(regs=="L=4 P=2"))
# qd <- setCNVoffsets(qd, cn)
```

---

summarizeScores *Subtract scores of different samples.*

Description

Based on a design matrix, scores matrices are subtracted, and a new ScoresList is returned, with the scores of the contrasts in it.

Usage

```
  ## S4 method for signature 'ScoresList,matrix'
summarizeScores(scores.list, design, verbose = TRUE)
```

Arguments

scores.list	A ScoresList object describing the coverage or intensity scores of a set of samples.
design	A matrix that contains only -1, 0, or 1.
verbose	Whether to print a statement explaining the function was called.

Value

A ScoresList object holding the scores of the contrasts that were specified by the design matrix.

Author(s)

Dario Strbenac

Examples

```
data(chr21genes)
data(samplesList)  # Loads 'samples.list.subset'.

fs <- featureScores(samples.list.subset[1:2], chr21genes, up = 2000, down = 1000,
                    freq = 500, s.width = 500)
d.matrix <- matrix(c(-1, 1))
colnames(d.matrix) <- "IP-input"
summarizeScores(fs, d.matrix)
```

---

writeWig *Writes sequencing data out into wiggle files*

Description

Writes sequencing data out into wiggle files

Usage

```
  ## S4 method for signature 'AffymetrixCelSet'
writeWig(rs, design=NULL, log2.adj=TRUE, verbose=TRUE)
  ## S4 method for signature 'GRangesList'
writeWig(rs, seq.len = NULL, design=NULL, sample=20, drop.zero=TRUE, normalize=TRUE, verbose=TRUE)
```

Arguments

rs	The sequencing or array data.
design	design matrix specifying the contrast to compute (i.e. the samples to use and what differences to take)
log2.adj	whether to take log2 of array intensities.
verbose	Whether to write progress to screen
seq.len	If sequencing reads need to be extended, the fragment size to be used
sample	At what basepair resolution to sample the genome at
drop.zero	Whether to write zero values to the wiggle file - TRUE saves diskspace
normalize	Whether to normalize each lane to its total number of reads, TRUE is suggested

Details

A wiggle file is created for each column in the design matrix (if design is left as NULL, then a file is created for each array/lane of sequencing). The filenames are given by the column names of the design matrix, and if ending in "gz" will be written out as a gzfile.

Value

Wiggle file(s) are created

Author(s)

Aaron Statham

Examples

```
#See examples in the manual
```

Index