Package ‘BRGenomics’

December 16, 2019

Type Package

Title Tools for the Efficient Analysis of High-Resolution Genomics Data

Version 0.2.0

Description This package provides useful and efficient utilites for the
analysis of high-resolution genomic data using standard Bioconductor
methods and classes.

License Artistic-2.0

Encoding UTF-8

LazyData FALSE

RoxygenNote 7.0.2

Depends GenomicRanges,
GenomicFeatures,
BiocParallel

Imports rtracklayer,
stats4,
BiocGenerics,
parallel,
S4Vectors,
IRanges,
AnnotationDbi,
GenomeInfoDb

Suggests BiocStyle,
knitr,
rmarkdown,
TxDb.Hsapiens.UCSC.hg38.knownGene,
TxDb.Hsapiens.UCSC.hg19.knownGene,
TxDb.Mmusculus.UCSC.mm10.knownGene,
TxDb.Mmusculus.UCSC.mm9.knownGene,
TxDb.Dmelanogaster.UCSC.dm6.ensGene,
TxDb.Dmelanogaster.UCSC.dm3.ensGene

Enhances DESeq2

biocViews Software,
DataImport,
PROseq,
RNAseq,
ATACseq,

NETseq,
ChIPseq,
CutAndRun,
Transcription,
GeneRegulation,
Normalization

VignetteBuilder knitr

R topics documented:

`binNDimensions` *N-dimensional binning of data by quantiles*

Description

This function takes in data along 1 or more dimensions, and for each dimension evenly divides the data in evenly-sized quantiles from the minimum to the maximum value. For each input data point, indices are returned giving its bin.

Usage

```
binNDimensions(..., quantiles = 10)
```

Arguments

`...`	Input data can be a single dataframe or any number of lists or vectors containing different measurements across the same samples.
`quantiles`	Either a number giving the number of quantiles to use for all dimensions (default = 10), or a vector containing the number of quantiles to use for each dimension of input data given.

Value

A dataframe containing indices in `1:quantiles` for each datapoint in each dimension.

`genebodies` *Extract Genebodies*

Description

Analagous to `GenomicRanges::promoters`, this function returns ranges that start and end downstream of the TSS. When `fix = "start"`, the function behaves differently depending on the sign of the end parameter. Currently, there's no way to set the ranges to continue a fixed distance past the original end sites when `fix = "start"`.

Usage

```
genebodies(
  genelist,
  start = 300,
  end = -300,
  fix_start = "start",
  fix_end = "end",
  min_window = 500
)
```

Arguments

`genelist`	A GRanges object containing genes of interest.
`start`	When `fix = "start"`, the distance downstream of the TSS where the new ranges should begin.
`end`	Where the ranges should end. When `end = 0`, the returned ranges keep the original end sites. When `end < 0`, the new ranges will end `abs(end)` number of bases before the original end site. When `end > 0` and `fix = "start"`, the new ends are fixed to `end` number of bases from the *original* start site.
`min.window`	The minimum size of a returned genebody length, after accounting for `start` and `end` parameters.
`fix`	If set to "end", function will return ranges centered around the end of the ranges. Negative values for `start` will begin the output ranges a fixed distance before the end of the input ranges; positive values will begin the ranges after the ends of the input ranges. The behavior of `end` is the same as for `start`, and errors will be returned if `end < start`.

Value

A GRanges object that may be shorter than `genelist` due to loss of short ranges.

`getCountsByPositions` *Get signal count matrix within regions of interest*

Description

Get signal count matrix within regions of interest

Usage

```
getCountsByPositions(
  dataset.gr,
  regions.gr,
  binsize = 1,
  field = "score",
  remove_empty = FALSE
)
```

Arguments

`dataset.gr`	A GRanges object in which signal is contained in metadata (typically in the "score" field).
`regions.gr`	A GRanges object containing all the regions of interest. All ranges must have the same width!
`binsize`	Size of bins (in bp) to use for counting within each range of `regions.gr`. Note that counts will *not* be length-normalized.
`field`	The metadata field of `dataset.gr` to be counted.
`remove_empty`	Logical indicating whether regions without signal should be removed.

Value

A matrix containing a row for each range in regions.gr, and a column for each bin.

`getCountsByRegions` *Signal counts in regions of interest*

Description

Count signal (e.g. read coverage) of data in each region of interest.

Usage

```
getCountsByRegions(dataset.gr, regions.gr, field = "score")
```

Arguments

`dataset.gr`	A GRanges object in which signal is contained in metadata (typically in the "score" field).
`regions.gr`	A GRanges object containing all the regions of interest.
`field`	The metadata field of `dataset.gr` to be counted.

Value

Returns a vector the same length as `regions.gr` containing signal found in each range.

`getMaxPositions` *Find sites with max signal in regions of interest*

Description

For each signal-containing region of interest, find the single site with the most signal. Sites can be found at base-pair resolution, or defined for larger bins.

Usage

```
getMaxPositions(
  regions.gr,
  dataset.gr,
  binsize = 1,
  field = "score",
  keep.score = F
)
```

Arguments

`regions.gr`	A GRanges object containing regions of interest.
`dataset.gr`	A GRanges object in which signal is contained in metadata (typically in the "score" field).
`binsize`	The size of bin in which to calculate signal scores.
`field`	The metadata field of `dataset.gr` to be counted.
`keep.score`	Logical indicating if the signal value at the max site should be reported. If set to `TRUE`, the values are kept as a new metadata column in `regions.gr`.

Value

Output is a GRanges object with regions.gr metadata, but each range only contains the site within each `regions.gr` range that had the most signal. If `binsize != 1`, a single site is still returned, but its position is set to the center of the bin. If the binsize is even, the site is rounded to be closer to the beginning of the range. If `keep.score = TRUE`, then the output will also have metadata for score at the max site. The output is *not* necessarily same length as `regions.gr`, as regions without signal are not returned. If *no regions* have signal (e.g. as could happen if running this function on a single region), the function will return an empty GRanges object with intact metadata columns.

getPausingIndices *Calculate pausing indices from user-supplied promoters & genebodies*

Description

Pausing index (PI) is calculated for each gene (within matched `promoters.gr` and `genebodies.gr`) as promoter signal counts divided by genebody signal counts. If `length.normalize = TRUE` (recommended), the signal counts within each range in `promoters.gr` and `genebodies.gr` are divided by their respective range widths (region lengths) before pausing indices are calculated.

Usage

```
getPausingIndices(
  dataset.gr,
  promoters.gr,
  genebodies.gr,
  field = "score",
  length_normalize = TRUE,
  remove_empty = FALSE
)
```

Arguments

`dataset.gr`	A GRanges object in which signal is contained in metadata (typically in the "score" field).
`promoters.gr`	A GRanges object containing all the regions of interest. The sum of all signal counts within is the pause index numerator.
`genebodies.gr`	A GRanges object containing all the regions of interest. The sum of all signal counts within is the pause index denominator.
`field`	The metadata field of `dataset.gr` to be counted, i.e. that contains the read-counts of interest.
`length_normalize`	A logical indicating if signal counts within regions of interest should be length normalized. The default is TRUE, which is recommended, especially if input regions don't all have the same width.
`remove_empty`	A logical indicating if genes without any signal should be removed. The default is FALSE.

Value

A vector of length given by the length of the genelist (or possibly shorter if `remove_empty = TRUE`).

`getStrandedCoverage` *Get strand-specific coverage*

Description

Computes strand-specific coverage signal, and returns a GRanges object with signal in the "score" metadata column. Function also works for non-strand-specific data.

Usage

```
getStrandedCoverage(gr, field = "score")
```

Arguments

`field`	The name of the field that contains readcounts. If no metadata field contains readcounts, and each range represents a single read, set to NULL.
`dataset.gr`	A GRanges object either containing ranges for each read, or one in which readcounts for individual ranges are contained in metadata (typically in the "score" field).

`import.bw_trim` *Import bigWig files (general)*

Description

General function for importing a single bigWig file as a GRanges object. The added functionality over `rtracklayer::import.bw` is in trimming odd chromosomes.

Usage

```
import.bw_trim(
  file,
  genome = NULL,
  keep_X = TRUE,
  keep_Y = TRUE,
  keep_M = FALSE,
  keep_nonstandard = FALSE
)
```

Arguments

`file`	Path of a bigWig file (non-stranded).
`genome`	Optional string for UCSC reference genome, e.g. "hg38". If given, non-standard chromosomes are trimmed.
`keep_X`	Logical indicating whether the X chromosome should be kept.
`keep_Y`	Logical indicating whether the Y chromosome should be kept.
`keep_M`	Logical indicating whether mitochondrial chromosomes should be kept.

Value

Imports a GRanges object

import.CoPRO *Import CoPRO (or similar) bedGraph files*

Description

This function imports plus/minus pairs of bedGraph files. This function is useful for when both 5'- and 3'-end information is to be maintained for each sequenced molecule.

Usage

```
import.CoPRO(
  plus_file,
  minus_file,
  genome = NULL,
  keep_X = TRUE,
  keep_Y = TRUE,
  keep_M = FALSE,
  keep_nonstandard = FALSE
)
```

Arguments

`genome`	Optional string for UCSC reference genome, e.g. "hg38". If given, non-standard chromosomes are trimmed.
`keep_X`	Logical indicating whether the X chromosome should be kept.
`keep_Y`	Logical indicating whether the Y chromosome should be kept.
`keep_M`	Logical indicating whether mitochondrial chromosomes should be kept.
`plus.file`	Path of plus strand bedGraph file.
`minus.file`	Path of minus strand bedGraph file.

Value

Imports a GRanges object containing entire strand-specific reads. Each range is unique, and the `score` metadata column indicates the number of identical reads (which share the same 5' and 3' ends).

Examples

```
md.import.CoPRO("~/LacZ_RNAi_plus.bedGraph",
"~/LacZ_RNAi_minus.bedGraph", "dm6")
```

`import.PROseq` *Import PRO-seq (or similar) bigWig files*

Description

This function imports plus/minus pairs of bigWig files containing basepair-resolution data, e.g. PRO-seq or PRO-cap data.

Usage

```
import.PROseq(
  plus_file,
  minus_file,
  genome = NULL,
  keep_X = TRUE,
  keep_Y = TRUE,
  keep_M = FALSE,
  keep_nonstandard = FALSE
)
```

Arguments

`genome`	Optional string for UCSC reference genome, e.g. "hg38". If given, non-standard chromosomes are trimmed, and options for sex and mitochondrial chromosomes are applied.
`keep_X`	Logical indicating whether the X chromosome should be kept.
`keep_Y`	Logical indicating whether the Y chromosome should be kept.
`keep_M`	Logical indicating whether mitochondrial chromosomes should be kept.
`plus_bw`	Path of plus strand bigWig file.
`minus_bw`	Path of minus strand bigWig file.

Value

Imports a GRanges object containing base-pair resolution data, with the `score` metadata column indicating readcounts at each base. All ranges are of width = 1.

Examples

```
md.import.PROseq("~/LacZ_RNAi_plus.bw", "~/LacZ_RNAi_minus.bw", "dm6")
```

`importGenesUCSC` *Import genes from UCSC*

Description

Imports all annotated genes by calling `GenomicFeatures::genes`, which provides a single range for each annotated gene. Currently supports hg38, hg19, mm10, mm9, dm6, and dm3. This script filters non-standard and mitochondrial chromosomes.

Usage

```
importGenesUCSC(
  genome,
  keep_X = TRUE,
  keep_Y = TRUE,
  keep_M = FALSE,
  keep_nonstandard = FALSE
)
```

Arguments

`genome`	A string indicating the UCSC reference genome, e.g. "hg38".
`keep_X`	A logical indicating if X chromosome transcripts should be kept.
`keep_Y`	A logical indicating if Y chromosome transcripts should be kept.

Value

A GRanges object, including metadata for unique identifiers.

See Also

`md.import.txs.ucsc`

`importTxsUCSC` *Import transcripts from UCSC*

Usage

```
importTxsUCSC(
  genome,
  keep_X = TRUE,
  keep_Y = TRUE,
  keep_M = FALSE,
  keep_nonstandard = FALSE
)
```

Arguments

`genome`	A string indicating the UCSC reference genome, e.g. "hg38".
`keep_X`	A logical indicating if X chromosome transcripts should be kept.
`keep_Y`	A logical indicating if Y chromosome transcripts should be kept.

Value

A GRanges object, including metadata for unique identifiers.

See Also

`md.import.genes.ucsc`

`makeGRangesBPres` *Make base-pair resolution GRanges object*

Description

Splits up all ranges in `gr` to be each 1 basepair wide. All information is preserved, including all metadata. To wit, `length(output.gr)` = `sum(width(dataset.gr))`.

Usage

```
makeGRangesBPres(gr)
```

Arguments

`gr`	A disjoint GRanges object.

`mergeGRangesData` *Merge base-pair resolution GRanges objects*

Description

Merges 2 or more GRanges objects. For each object, the range widths must all be 1, and the `score` metadata column contains coverage information at each site.

Usage

```
mergeGRangesData(..., field = "score", ncores = detectCores())
```

Arguments

`...`	One or more additional GRanges objects also fitting the above description.
`dataset.gr`	A GRanges object fitting the above description.

Value

A single GRange object containing all sites of the input objects, and the sum of all scores at all sites.

See Also

`md.import.bigWigs`

`metaSubsample` *Iterative Subsampling for Metaplotting*

Description

Iterative Subsampling for Metaplotting

Usage

```
metaSubsample(
  dataset.gr,
  regions.gr,
  binsize = 1,
  first_output_xval = 1,
  sample_name = deparse(substitute(dataset.gr)),
  n_iter = 1000,
  prop_subsample = 0.1,
  lower = 0.125,
  upper = 0.875,
  NF = 1,
  field = "score",
  remove_empty = FALSE,
  ncores = 1
)
```

Arguments

`dataset.gr`	A GRanges object in which signal is contained in metadata (typically in the "score" field).
`regions.gr`	A GRanges object containing intervals over which to metaplot. All ranges must have the same width.
`binsize`	The size of bin (number of columns, e.g. basepairs) to use for metaplotting. Especially important for metaplots over large/sparse regions.
`first_output_xval`	The relative start position of the first bin, e.g. if regions.gr begins at 50 bases upstream of the TSS, set `first_output_xval = -50`. This number only affects the x-values that are returned, which are provided as a convenience.
`sample_name`	Defaults to the name of `dataset.gr`.
`n_iter`	Number of random subsampling iterations to perform. Default is 1000.
`prop_subsample`	The proportion of the ranges in regions.gr (e.g. the proportion of genes) to subsample in each iteration. The default is 0.1.
`lower`	The lower quantile of subsampled signal means to return. The default is 0.125 (12.5th percentile).
`upper`	The upper quantile of subsampled signal means to return. The default is 0.875 (85.5th percentile).

`NF`	Optional normalization factor by which to multiply the counts.
`field`	The metadata field of `dataset.gr` to be counted.
`remove_empty`	A logical indicating whether regions without signal should be removed from the analysis.
`ncores`	Number of cores to use for parallel computation. As of writing, parallel processing doesn't show any benefit for short computation times (e.g. <1 minute for our typical experience on a laptop).

Value

Dataframe containing x-values, means, lower quantiles, upper quantiles, and the sample name (as a convenience for row-binding multiple of these dataframes).

`metaSubsampleMatrix` *Iterative Subsampling for Metaplotting (On Count Matrices)*

Description

In the most general sense, this function performs iterations of randomly subsampling rows of a matrix, and returns a summary of mean values calculated for each column. The typical application is for generating metaplots, with the typical input being a matrix in which each row is a gene or other region of interest, each column is a position within that gene (either a specific basepair or a bin), and element values are signal (e.g. read counts) within those positions.

Usage

```
metaSubsampleMatrix(
  counts.mat,
  binsize = 1,
  first_output_xval = 1,
  sample_name = deparse(substitute(dataset.gr)),
  n_iter = 1000,
  prop_subsample = 0.1,
  lower = 0.125,
  upper = 0.875,
  NF = 1,
  ncores = 1
)
```

Arguments

`counts.mat`	A matrix of signal counts in which rows are regions of interest and columns are sites/bins in each region.
`binsize`	The size of bin (number of columns, e.g. basepairs) to use for metaplotting. Especially important for metaplots over large/sparse regions.
`first_output_xval`	The relative start position of the first bin, e.g. if regions.gr begins at 50 bases upstream of the TSS, set `first_output_xval = -50`. This number only affects the x-values that are returned, which are provided as a convenience.
`sample_name`	Defaults to the name of `dataset.gr`.

`n_iter`	Number of random subsampling iterations to perform. Default is 1000.
`prop_subsample`	The proportion of rows to subsample in each iteration. The default is 0.1.
`lower`	The lower quantile of subsampled signal means to return. The default is 0.125 (12.5th percentile).
`upper`	The upper quantile of subsampled signal means to return. The default is 0.875 (85.5th percentile).
`NF`	Optional normalization factor by which to multiply the counts.
`ncores`	Number of cores to use for parallel computation. As of writing, parallel processing doesn't show any benefit for short computation times (e.g. <1 minute for our typical experience on a laptop).

Value

Dataframe containing x-values, means, lower quantiles, upper quantiles, and the sample name (as a convenience for row-binding multiple of these dataframes).

`metaSubsampleScaled` *Iterative Subsampling for Metaplotting (With Scaled Regions)*

Description

This function can perform iterative metaplot subsampling in several configurations. All arguments for regions-of-interest for metaplot subsampling are optional, as long as at least one region is supplied. Unnamed arguments are regions-of-interest of variable "widths" (i.e. "lengths" in basepairs) over which to perform length-scaled metaplot subsampling. Length-scaled metaplot subsampling involves dividing each range (e.g. each region-of-interest) into `nbins_scaled` number of equally-sized bins, and obtaining signal counts in each bin, divided by the size of the bin for that particular region-of-interest. Non-length-scaled subsampling (as would be done using `md.meta.subsample`) is performed on the named arguments `linear_regions_start.gr` and `linear_regions_end.gr`. The output is constructed in the order `linear_regions_start.gr`, unnamed scaled regions (in order given), and then `linear_regions_end.gr`, with x-values corresponding to the bin number.

Usage

```
metaSubsampleScaled(
  dataset.gr,
  ...,
  linear_regions_start.gr = NULL,
  linear_regions_end.gr = NULL,
  nbins_scaled = 100,
  nbins_linear_start = unique(width(linear_regions_start.gr)),
  nbins_linear_end = unique(width(linear_regions_end.gr)),
  sample_name = deparse(substitute(dataset.gr)),
  n_iter = 1000,
  prop_subsample = 0.1,
  lower = 0.125,
  upper = 0.875,
  NF = 1,
  field = "score",
  remove_empty = FALSE,
  ncores = 1
)
```

Arguments

`dataset.gr`	A GRanges object in which signal is contained in metadata (typically in the "score" field).
`...`	0 or more GRanges objects containing regions of interest over which to do length-scaled signal counting and metaplot subsampling. The output x-positions will be determined by the order in which these regions are supplied, the number of bins used for counting signal within variable length regions (`nbins_scaled`), and whether or not a `linear_regions_start.gr` object is given.
`linear_regions_start.gr`	Optional GRanges object containing regions of interest over which to do linear (un-scaled) signal counting and metaplot subsampling. Because no length-scaling is performed, all ranges must have the same width. These regions will be put before any supplied regions for length-scaled metaplot subsampling, i.e. the first `nbins_linear_start` x-values will be from subsampling `linear_regions_start.gr`.
`linear_regions_end.gr`	Optional GRanges object containing regions of interest over which to do linear (un-scaled) signal counting and metaplot subsampling. Because no length-scaling is performed, all ranges must have the same width. These regions will be placed after any supplied regions for length-scaled metaplot subsampling, i.e. the last `nbins_linear_end` x-values will be from subsampling `linear_regions_end.gr`.
`nbins_scaled`	The number of bins to use for length scaling signal counts.
`nbins_linear_start`	The number of bins to use for counting signal within `linear_regions_start.gr`. Defaults to the width of the regions, i.e. a binsize of 1 (no binning).
`nbins_linear_end`	The number of bins to use for counting signal within `linear_regions_end.gr`. Defaults to the width of the regions, i.e. a binsize of 1 (no binning).
`sample_name`	Defaults to the name of `dataset.gr`.
`n_iter`	Number of random subsampling iterations to perform. Default is 1000.
`prop_subsample`	The proportion of the genelist (regions.gr) to subsample in each iteration. The default is 0.1.
`lower`	The lower quantile of subsampled signal means to return. The default is 0.125 (12.5th percentile).
`upper`	The upper quantile of subsampled signal means to return. The default is 0.875 (85.5th percentile).
`NF`	Optional normalization factor by which to multiply the counts.
`field`	The metadata field of `dataset.gr` to be counted.
`remove_empty`	A logical indicating whether regions without signal should be removed from the analysis.
`ncores`	Number of cores to use for parallel computation. As of writing, parallel processing doesn't show any benefit for short computation times (e.g. <1 minute for our typical experience on a laptop).

Details

The user must be able to determine the correct meaning of the bin numbers in the final output, and for that reason arguments for binning are always explicitly the number of bins, and not the size of the bins (as would be possible for linear (un-scaled) regions). For example, if the user provides `linear_regions_start.gr`, one unnamed GRanges for length-scaled subsampling, and `linear_regions_end.gr`, the output x-values `1:nbins_linear_start` will correspond to equally-sized bins in `linear_regions_start.gr`; the subsequent `nbins_scaled` x-values will correspond to variably-sized bins in the unnamed GRanges object given, and the final `nbins_linear_end` x-values will correspond to equally-sized bins in `linear_regions_end.gr`.

Value

Dataframe containing x-values, means, lower quantiles, upper quantiles, and the sample name (as a convenience for row-binding multiple output dataframes). X-values correspond to bins based on the input regions given and the specified binsizes to use.

Examples

```
md.meta.scaled_subsample(my_proseq_data, genes.early_genebodies,
genes.late_genebodies,
linear_regions_start.gr = genes.promoter_proximal,
linear_regions_end.gr = genes.cps_proximal,
nbins_scaled = 500, nbins_linear_end = 500)
```

`subsampleGRanges` *Randomly subsample reads from GRanges dataset*

Description

Currently only works if signal is integer

Usage

```
subsampleGRanges(dataset.gr, n = NULL, prop = NULL, field = "score")
```

Arguments

`dataset.gr`	A GRanges object in which signal (e.g. readcounts) are contained within metadata.
`n`	Number of reads to subsample. Either `n` or `prop` can be given.
`prop`	Proportion of total signal to subsample.
`field`	

subsetRegionsBySignal *Subset regions of interest by highest signal*

Description

Subsets regions based on signal in a dataset, taking only the top quantile of regions.

Usage

```
subsetRegionsBySignal(
  regions.gr,
  dataset.gr,
  regions_quantile,
  field = "score",
  order_by_rank = FALSE,
  density = FALSE
)
```

Arguments

`regions.gr`	A GRanges object containing regions of interest.
`dataset.gr`	A GRanges object in which signal is contained in metadata (typically in the "score" field).
`regions_quantile`	The proportion of regions.gr to return, e.g. if `regions_quantile = 0.2`, the top 20% of regions by signal are returned.
`field`	The metadata field of `dataset.gr` to be counted.
`order_by_rank`	Logical indicating if genes should be returned in order of their expression. If `FALSE` (the default), genes are sorted by their positions.
`density`	A logical indicating whether signal counts should be normalized to the width of ranges in `regions.gr`. By default, the function only considers the total signal in each range.

Value

A GRanges object of length `length(regions.gr) * regions_quantile`.

Index