Package 'GenomicDistributions'

October 24, 2025

Version 1.17.1

Date 2025-04-23

Title GenomicDistributions: fast analysis of genomic intervals with Bioconductor

Description If you have a set of genomic ranges, this package can help you with visualization and comparison. It produces several kinds of plots, for example: Chromosome distribution plots, which visualize how your regions are distributed over chromosomes; feature distance distribution plots, which visualizes how your regions are distributed relative to a feature of interest, like Transcription Start Sites (TSSs); genomic partition plots, which visualize how your regions overlap given genomic features such as promoters, introns, exons, or intergenic regions. It also makes it easy to compare one set of ranges to another.

Depends R (>= 4.0), IRanges, GenomicRanges

Imports data.table, ggplot2, reshape2, methods, utils, Biostrings, plyr, dplyr, scales, broom, GenomeInfoDb, stats

Suggests AnnotationFilter, rtracklayer, testthat, knitr, BiocStyle, rmarkdown, GenomicDistributionsData

Enhances BSgenome, extrafont, ensembldb, GenomicFeatures

LazyData true

VignetteBuilder knitr

License BSD_2_clause + file LICENSE

biocViews Software, GenomeAnnotation, GenomeAssembly, DataRepresentation, Sequencing, Coverage, FunctionalGenomics, Visualization

RoxygenNote 7.3.2

URL http://code.databio.org/GenomicDistributions

BugReports http://github.com/databio/GenomicDistributions

Encoding UTF-8

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

git_branch devel

git_last_commit 981f02c

git_last_commit_date 2025-04-23

Repository Bioconductor 3.23

Date/Publication 2025-10-24

Author Kristyna Kupkova [aut, cre],
Jose Verdezoto [aut],
Tessa Danehy [aut],
John Lawson [aut],
Jose Verdezoto [aut],
Michal Stolarczyk [aut],
Jason Smith [aut],
Bingjie Xue [aut],
Sophia Rogers [aut],
John Stubbs [aut],
Nathan C. Sheffield [aut]

Maintainer Kristyna Kupkova <kristynakupkova@gmail.com>

Contents

`GenomicDistributions-package`

Produces summaries and plots of features distributed across genomes

---

Description

If you have a set of genomic ranges, the GenomicDistributions R package can help you with some simple visualizations. Currently, it can produce two kinds of plots: First, the chromosome distribution plot, which visualizes how your regions are distributed over chromosomes; and second, the feature distribution plot, which visualizes how your regions are distributed relative to a feature of interest, like Transcription Start Sites (TSSs).

Author(s)

Maintainer: Kristyna Kupkova <kristynakupkova@gmail.com>

Authors:

- Jose Verdezoto
- Tessa Danehy
- John Lawson
- Jose Verdezoto
- Michal Stolarczyk
- Jason Smith
- Bingjie Xue
- Sophia Rogers
- John Stubbs
- Nathan C. Sheffield <nathan@code.databio.org>

See Also

Useful links:

- http://code.databio.org/GenomicDistributions
- Report bugs at http://github.com/databio/GenomicDistributions

---

`.requireAndReturn` *Checks to make sure a package object is installed, and if so, returns it. If the library is not installed, it issues a warning and returns NULL.*

Description

Checks to make sure a package object is installed, and if so, returns it. If the library is not installed, it issues a warning and returns NULL.

Usage

```
.requireAndReturn(BSgenomeString)
```

Arguments

BSgenomeString A BSgenome compatible genome string.

Value

A BSgenome object if installed.

.validateInputs *Checks class of the list of variables. To be used in functions*

Description

Checks class of the list of variables. To be used in functions

Usage

```
.validateInputs(checkList)
```

Arguments

checkList	list of object to check, e.g. list(varname=c("data.frame", "numeric")). Multiuple strings in the vector are treated as OR.

Value

A warning if the wrong input class is provided.

Examples

```
x = function(var1) {
    cl = list(var1=c("numeric","character"))
    .validateInputs(cl)
    return(var1^2)
}
```

binBSGenome *Bins a BSgenome object.*

Description

Given a BSgenome object (to be loaded via `loadBSgenome`), and a number of bins, this will bin that genome. It is a simple wrapper of the `binChroms` function

Usage

```
binBSGenome(genome, binCount)
```

Arguments

`genome`	A UCSC-style string denoting reference assembly (e.g. 'hg38')
`binCount`	number of bins per chromosome

Value

A data.table object showing the region and bin IDs of the reference genome.

Examples

```
## Not run:
binCount = 1000
refGenomeBins = binBSGenome("hg19", binCount)

## End(Not run)
```

binChroms *Naively splits a chromosome into bins*

Description

Given a list of chromosomes with corresponding sizes, this script will produce (roughly) evenly-sized bins across the chromosomes. It does not account for assembly gaps or the like.

Usage

```
binChroms(binCount, chromSizes)
```

Arguments

`binCount`	number of bins (total; *not* per chromosome)
`chromSizes`	a named list of size (length) for each chromosome.

Value

A data.table object assigning a bin ID to each chromosome region.

Examples

```
chromSizes = c(chr1=249250621, chr2=243199373, chr3=198022430)
cBins = binChroms(1000, chromSizes)
```

`binRegion` *Divide regions into roughly equal bins*

Description

Given a start coordinate, end coordinate, and number of bins to divide, this function will split the regions into that many bins. Bins will be only approximately the same size, due to rounding. (they should not be more than 1 different).

Usage

```
binRegion(start, end, binSize = NULL, binCount = NULL, indicator = NULL)
```

Arguments

`start`	The starting coordinate
`end`	The ending coordinate
`binSize`	The size of bin to divide the genome into. You must supply either binSize (priority) or binCount.
`binCount`	The number of bins to divide. If you do not supply binSize, you must supply binCount, which will be used to calculate the binSize.
`indicator`	A vector with identifiers to keep with your bins, in case you are doing this on a long table with multiple segments concatenated

Details

Use case: take a set of regions, like CG islands, and bin them; now you can aggregate signal scores across the bins, giving you an aggregate signal in bins across many regions of the same type.

In theory, this just runs on 3 values, but you can run it inside a data.table j expression to divide a bunch of regions in the same way.

Value

A data.table, expanded to nrow = number of bins, with these id columns: id: region ID binID: repeating ID (this is the value to aggregate across) ubinID: unique bin IDs

Examples

```
Rbins = binRegion(1, 3000, 100, 1000)
```

`BSdtToGRanges` *Converts a list of data.tables (From BSreadbeds) into GRanges.*

Description

Converts a list of data.tables (From BSreadbeds) into GRanges.

Usage

```
BSdtToGRanges(dtList)
```

Arguments

`dtList`	A list of data.tables

Value

A GRangesList object.

`calcChromBins` *Calculates the distribution of a query set over the genome*

Description

Returns a data.table showing counts of regions from the query that overlap with each bin. In other words, where on which chromosomes are the ranges distributed? You must provide binned regions. Only the midpoint of each query region is used to test for overlap with the bin regions.

Usage

```
calcChromBins(query, bins)
```

Arguments

`query`	A GenomicRanges or GenomicRangesList object with query regions
`bins`	Pre-computed bins (as a GRangesList object) to aggregate over; for example, these could be genome bins

Value

A data.table showing where on which chromosomes ranges are distributed.

Examples

```
chromSizes = getChromSizes("hg19")
genomeBins  = getGenomeBins(chromSizes)
chromDistribution = calcChromBins(vistaEnhancers, genomeBins)

vistaSftd = GenomicRanges::shift(vistaEnhancers, 100000)
vistaSftd2 = GenomicRanges::shift(vistaEnhancers, 200000)
calcChromBins(vistaEnhancers, GRangesList(vistaSftd, vistaSftd2))
```

`calcChromBinsRef` *Returns the distribution of query over a reference assembly Given a query set of elements (a GRanges object) and a reference assembly (*e.g. 'hg38'), this will aggregate and count the distribution of the query elements across bins of the reference genome. This is a helper function to create features for common genomes. It is a wrapper of* `calcChromBins`*, which is more general.*

Description

Returns the distribution of query over a reference assembly Given a query set of elements (a GRanges object) and a reference assembly (*e.g. 'hg38'), this will aggregate and count the distribution of the query elements across bins of the reference genome. This is a helper function to create features for common genomes. It is a wrapper of `calcChromBins`, which is more general.

Usage

```
calcChromBinsRef(query, refAssembly, binCount = 3000)
```

Arguments

`query`	A GenomicRanges or GenomicRangesList object with query regions
`refAssembly`	A character vector that will be used to grab chromosome sizes with `getChromSizes`
`binCount`	Number of bins to divide the chromosomes into

Value

A data.table showing the distribution of regions across bins of the reference genome.

Examples

```
ChromBins = calcChromBinsRef(vistaEnhancers, "hg19")
```

`calcChromBinsRefSlow` *Returns the distribution of query over a reference assembly Given a query set of elements (a GRanges object) and a reference assembly (*e.g. 'hg38'), this will aggregate and count the distribution of the query elements across bins of the reference genome. This is a helper function to create features for common genomes. It is a wrapper of* `calcChromBins`*, which is more general.*

Description

Returns the distribution of query over a reference assembly Given a query set of elements (a GRanges object) and a reference assembly (*e.g. 'hg38'), this will aggregate and count the distribution of the query elements across bins of the reference genome. This is a helper function to create features for common genomes. It is a wrapper of `calcChromBins`, which is more general.

Usage

```
calcChromBinsRefSlow(query, refAssembly, binCount = 3000)
```

Arguments

`query`	A GenomicRanges or GenomicRangesList object with query regions
`refAssembly`	A character vector that will be used to grab chromosome sizes with `getChromSizes`
`binCount`	Number of bins to divide the chromosomes into

Value

A data.table showing the distribution of regions across bins of the reference genome.

Examples

```
ChromBins = calcChromBinsRef(vistaEnhancers, "hg19")
```

`calcCumulativePartitions`

Calculates the cumulative distribution of overlaps between query and arbitrary genomic partitions

Description

Takes a GRanges object, then assigns each element to a partition from the provided partitionList, and then tallies the number of regions assigned to each partition. A typical example of partitions is promoter, exon, intron, etc; this function will yield the number of each for a query GRanges object There will be a priority order to these, to account for regions that may overlap multiple genomic partitions.

Usage

```
calcCumulativePartitions(query, partitionList, remainder = "intergenic")
```

Arguments

query	GRanges or GRangesList with regions to classify.
partitionList	An ORDERED and NAMED list of genomic partitions GRanges. This list must be in priority order; the input will be assigned to the first partition it overlaps.
remainder	Which partition do you want to account for 'everything else'?

Value

A data.frame assigning each element of a GRanges object to a partition from a previously provided partitionList.

Examples

```
partitionList = genomePartitionList(geneModels_hg19$genesGR,
                                    geneModels_hg19$exonsGR,
                                    geneModels_hg19$threeUTRGR,
                                    geneModels_hg19$fiveUTRGR)
calcCumulativePartitions(vistaEnhancers, partitionList)
```

---

calcCumulativePartitionsRef

Calculates the cumulative distribution of overlaps for a query set to a reference assembly

---

Description

This function is a wrapper for `calcCumulativePartitions` that uses built-in partitions for a given reference genome assembly.

Usage

```
calcCumulativePartitionsRef(query, refAssembly)
```

Arguments

query	A GenomicRanges or GenomicRangesList object with query regions
refAssembly	A character vector specifying the reference genome assembly (*e.g.* 'hg19'). This will be used to grab chromosome sizes with `getTSSs`.

Value

A data.frame indicating the number of query region overlaps in several genomic partitions.

Examples

```
calcCumulativePartitionsRef(vistaEnhancers, "hg19")
```

`calcDinuclFreq` *Calculate Dinuclotide content over genomic ranges*

Description

Given a reference genome (BSgenome object) and ranges on the reference, this function returns a data.table with counts of dinucleotides within the GRanges object.

Usage

```
calcDinuclFreq(query, ref, rawCounts = FALSE)
```

Arguments

`query`	A GRanges object with query sets
`ref`	Reference genome BSgenome object
`rawCounts`	a logical indicating whether the raw numbers should be displayed, rather than percentages (optional).

Value

A data.table with counts of dinucleotides across the GRanges object

Examples

```
## Not run:
bsg = loadBSgenome('hg19')
DNF = calcDinuclFreq(vistaEnhancers, bsg)

## End(Not run)
```

`calcDinuclFreqRef` *Calculate dinucleotide content over genomic ranges*

Description

Given a reference genome (BSgenome object) and ranges on the reference, this function returns a data.table with counts of dinucleotides within the GRanges object.

Usage

```
calcDinuclFreqRef(query, refAssembly, rawCounts = FALSE)
```

Arguments

query	A GRanges object with query sets
refAssembly	A character vector specifying the reference genome assembly (*e.g.* 'hg19'). This will be used to grab chromosome sizes with getTSSs.
rawCounts	a logical indicating whether the raw numbers should be displayed, rather than percentages (optional).

Value

A numeric vector or list of vectors with the GC percentage of the query regions.

Examples

```
## Not run:
query = system.file("extdata", "vistaEnhancers.bed.gz", package="GenomicDistributions")
GRquery = rtracklayer::import(query)
refAssembly = 'hg19'
DNF = calcDinuclFreqRef(GRquery, refAssembly)

## End(Not run)
```

---

calcExpectedPartitions

Calculates expected partiton overlap based on contribution of each feature (partition) to genome size. Expected and observed overlaps are then compared.

Description

Calculates expected partiton overlap based on contribution of each feature (partition) to genome size. Expected and observed overlaps are then compared.

Usage

```
calcExpectedPartitions(
  query,
  partitionList,
  genomeSize = NULL,
  remainder = "intergenic",
  bpProportion = FALSE
)
```

Arguments

query	GRanges or GRangesList with regions to classify.
partitionList	An ORDERED (if bpProportion=FALSE) and NAMED list of genomic partitions GRanges. This list must be in priority order; the input will be assigned to the first partition it overlaps. However, if bpProportion=TRUE, the list does not need ordering.
genomeSize	The number of bases in the query genome. In other words, the sum of all chromosome sizes.
remainder	Which partition do you want to account for 'everything else'?
bpProportion	logical indicating if overlaps should be calculated based on number of base pairs overlapping with each partition. bpProportion=FALSE does overlaps in priority order, bpProportion=TRUE counts number of overlapping base pairs between query and each partition.

Value

A data.frame assigning each element of a GRanges object to a partition from a previously provided partitionList.The data.frame also contains Chi-square p-values calculated for observed/expected overlaps on each individual partition.

Examples

```
partitionList = genomePartitionList(geneModels_hg19$genesGR,
                                    geneModels_hg19$exonsGR,
                                    geneModels_hg19$threeUTRGR,
                                    geneModels_hg19$fiveUTRGR)
chromSizes = getChromSizes('hg19')
genomeSize = sum(chromSizes)
calcExpectedPartitions(vistaEnhancers, partitionList, genomeSize)
```

---

calcExpectedPartitionsRef

Calculates the distribution of observed versus expected overlaps for a query set to a reference assembly

---

Description

This function is a wrapper for `calcExpectedPartitions` that uses built-in partitions for a given reference genome assembly.

Usage

```
calcExpectedPartitionsRef(query, refAssembly, bpProportion = FALSE)
```

Arguments

query	A GenomicRanges or GenomicRangesList object with query regions
refAssembly	A character vector specifying the reference genome assembly (*e.g.* 'hg19'). This will be used to grab annotation models with getGeneModels, and chromosome sizes withgetChromSizes
bpProportion	logical indicating if overlaps should be calculated based on number of base pairs overlapping with each partition. bpProportion=FALSE does overlaps in priority order, bpProportion=TRUE counts number of overlapping base pairs between query and each partition.

Value

A data.frame indicating the number of query region overlaps in several genomic partitions.

Examples

```
calcExpectedPartitionsRef(vistaEnhancers, "hg19")
```

---

calcFeatureDist *Find the distance to the nearest genomic feature*

Description

For a given query set of genomic regions, and a given feature set of regions, this function will return the distance for each query region to its closest feature. It ignores strand and returns the distance as positive or negative, depending on whether the feature is upstream or downstream

Usage

```
calcFeatureDist(query, features)
```

Arguments

query	A GRanges or GRangesList object with query sets
features	A GRanges object with features to test distance to

Details

This function is similar to the bioconductor distanceToNearest function, but returns negative values for downstream distances instead of absolute values. This allows you to assess the relative location.

Value

A vector of genomic distances for each query region relative to its closest feature.

Examples

```
vistaSftd = GenomicRanges::shift(vistaEnhancers, 100000)
calcFeatureDist(vistaEnhancers, vistaSftd)
```

calcFeatureDistRefTSS *Calculates the distribution of distances from a query set to closest TSS*

Description

Given a query GRanges object and an assembly string, this function will grab the TSS list for the given reference assembly and then calculate the distance from each query feature to the closest TSS. It is a wrapper of `calcFeatureDist` that uses built-in TSS features for a reference assembly

Usage

```
calcFeatureDistRefTSS(query, refAssembly)
```

Arguments

`query`	A GenomicRanges or GenomicRangesList object with query regions
`refAssembly`	A character vector specifying the reference genome assembly (*e.g.* 'hg19'). This will be used to grab chromosome sizes with `getTSSs`.

Value

A vector of distances for each query region relative to TSSs.

Examples

```
calcFeatureDistRefTSS(vistaEnhancers, "hg19")
```

calcGCContent *Calculate GC content over genomic ranges*

Description

Given a reference genome as a BSgenome object and some ranges on that reference, this function will return a vector of the same length as the granges object, with percent of Cs and Gs.

Usage

```
calcGCContent(query, ref)
```

Arguments

`query`	A GenomicRanges or GenomicRangesList object with query regions.
`ref`	Reference genome BSgenome object.

Value

A numeric vector of list of vectors with the GC percentage of the query regions.

Examples

```
## Not run:
bsg = loadBSgenome('hg19')
gcvec = calcGCContent(vistaEnhancers, bsg)

## End(Not run)
```

`calcGCContentRef` *Calculate GC content over genomic ranges*

Description

Given a reference genome as a BSgenome object and some ranges on that reference, this function will return a vector of the same length as the granges object, with percent of Cs and Gs.

Usage

```
calcGCContentRef(query, refAssembly)
```

Arguments

`query`	A GenomicRanges or GenomicRangesList object with query regions
`refAssembly`	A character vector specifying the reference genome assembly (*e.g.* 'hg19'). This will be used to grab chromosome sizes with `getTSSs`.

Value

A numeric vector or list of vectors with the GC percentage of the query regions.

Examples

```
## Not run:
refAssembly = 'hg19'
GCcontent = calcGCContentRef(vistaEnhancers, refAssembly)

## End(Not run)
```

`calcNearestNeighbors` *Group regions from the same chromosome together and compute the distance of a region to its nearest neighbor. Distances are then lumped into a numeric vector.*

Description

Group regions from the same chromosome together and compute the distance of a region to its nearest neighbor. Distances are then lumped into a numeric vector.

Usage

```
calcNearestNeighbors(query, correctRef = "None")
```

Arguments

`query`	A GRanges or GRangesList object.
`correctRef`	A string indicating the reference genome to use if Nearest neighbor distances are corrected for the number of regions in a regionSet.

Value

A numeric vector or list of vectors containing the distance of regions to their nearest neighbors.

Examples

```
Nneighbors = calcNearestNeighbors(vistaEnhancers)
```

`calcNeighborDist` *Group regions from the same chromosome together and calculate the distances of a region to its upstream and downstream neighboring regions. Distances are then lumped into a numeric vector.*

Description

Group regions from the same chromosome together and calculate the distances of a region to its upstream and downstream neighboring regions. Distances are then lumped into a numeric vector.

Usage

```
calcNeighborDist(query, correctRef = "None")
```

Arguments

`query`	A GRanges or GRangesList object.
`correctRef`	A string indicating the reference genome to use if distances are corrected for the number of regions in a regionSet.

Value

A numeric vector or list with different vectors containing the distances of regions to their upstream/downstream neighbors.

Examples

```
dist = calcNeighborDist(vistaEnhancers)
```

`calcPartitions` *Calculates the distribution of overlaps between query and arbitrary genomic partitions*

Description

Takes a GRanges object, then assigns each element to a partition from the provided partitionList, and then tallies the number of regions assigned to each partition. A typical example of partitions is promoter, exon, intron, etc; this function will yield the number of each for a query GRanges object There will be a priority order to these, to account for regions that may overlap multiple genomic partitions.

Usage

```
calcPartitions(
  query,
  partitionList,
  remainder = "intergenic",
  bpProportion = FALSE
)
```

Arguments

`query`	GRanges or GRangesList with regions to classify
`partitionList`	an ORDERED (if bpProportion=FALSE) and NAMED list of genomic partitions GRanges. This list must be in priority order; the input will be assigned to the first partition it overlaps. bpProportion=TRUE, the list does not need ordering.
`remainder`	A character vector to assign any query regions that do not overlap with anything in the partitionList. Defaults to "intergenic"
`bpProportion`	logical indicating if overlaps should be calculated based on number of base pairs overlapping with each partition. bpProportion=FALSE does overlaps in priority order, bpProportion=TRUE counts number of overlapping base pairs between query and each partition.

Value

A data.frame assigning each element of a GRanges object to a partition from a previously provided partitionList.

Examples

```
partitionList = genomePartitionList(geneModels_hg19$genesGR,
                                    geneModels_hg19$exonsGR,
                                    geneModels_hg19$threeUTRGR,
                                    geneModels_hg19$fiveUTRGR)
calcPartitions(vistaEnhancers, partitionList)
```

`calcPartitionsRef` *Calculates the distribution of overlaps for a query set to a reference assembly*

Description

This function is a wrapper for `calcPartitions` and `calcPartitionPercents` that uses built-in partitions for a given reference genome assembly.

Usage

```
calcPartitionsRef(query, refAssembly, bpProportion = FALSE)
```

Arguments

`query`	A GenomicRanges or GenomicRangesList object with query regions
`refAssembly`	A character vector specifying the reference genome assembly (*e.g.* 'hg19'). This will be used to grab annotation models with `getGeneModels`
`bpProportion`	logical indicating if overlaps should be calculated based on number of base pairs overlapping with each partition. bpProportion=FALSE does overlaps in priority order, bpProportion=TRUE counts number of overlapping base pairs between query and each partition.

Value

A data.frame indicating the number of query region overlaps in several genomic partitions.

Examples

```
calcPartitionsRef(vistaEnhancers, "hg19")
```

calcSummarySignal *The function calcSummarySignal takes the input BED file(s) in form of GRanges or GRangesList object, overlaps it with all defined open chromatin regions across conditions (e.g. cell types) and returns a matrix, where each row is the input genomic region (if overlap was found), each column is a condition, and the value is a meam signal from regions where overlap was found.*

Description

The function calcSummarySignal takes the input BED file(s) in form of GRanges or GRangesList object, overlaps it with all defined open chromatin regions across conditions (e.g. cell types) and returns a matrix, where each row is the input genomic region (if overlap was found), each column is a condition, and the value is a meam signal from regions where overlap was found.

Usage

```
calcSummarySignal(query, signalMatrix)
```

Arguments

query	Genomic regions to be analyzed. Can be GRanges or GRangesList object.
signalMatrix	Matrix with signal values in predfined regions, where rows are predefined genomic regions, columns are conditions (e.g. cell types in which the signal was measured). First column contains information about the genomic region in following form: chr_start_end. Can be either data.frame or data.table object.

Value

A list with named components: signalSummaryMatrix - data.table with cell specific open chromatin signal values for query regions matrixStats - data.frame containing boxplot stats for individual cell type

Examples

```
signalSummaryList = calcSummarySignal(vistaEnhancers, exampleOpenSignalMatrix_hg19)
```

calcWidth *Calculate the widths of regions*

Description

The length of a genomic region (the distance between the start and end) is called the width When given a query set of genomic regions, this function returns the width

Usage

```
calcWidth(query)
```

Arguments

query	A GRanges or GRangesList object with query sets

Value

A vector of the widths (end-start coordinates) of GRanges objects.

Examples

```
regWidths = calcWidth(vistaEnhancers)
```

---

cellTypeMetadata *Table the maps cell types to tissues and groups*

Description

Table the maps cell types to tissues and groups

Usage

```
data(cellTypeMetadata)
```

Format

data.table with 3 columns (cellType, tissue and group) and 74 rows (one per cellType)

Source

self-curated dataset

chromSizes_hg19 *hg19 chromosome sizes*

Description

A dataset containing chromosome sizes for Homo Sapiens hg38 genome assembly

Usage

```
data(chromSizes_hg19)
```

Format

A named vectors of lengths with one item per chromosome

Source

BSgenome.Hsapiens.UCSC.hg19 package

dtToGr *Converts a data.table (DT) object to a GenomicRanges (GR) object. Tries to be intelligent, guessing chr and start, but you have to supply end or other columns if you want them to be carried into the GR.*

Description

Converts a data.table (DT) object to a GenomicRanges (GR) object. Tries to be intelligent, guessing chr and start, but you have to supply end or other columns if you want them to be carried into the GR.

Usage

```
dtToGr(
  DT,
  chr = "chr",
  start = "start",
  end = NA,
  strand = NA,
  name = NA,
  splitFactor = NA,
  metaCols = NA
)
```

Arguments

DT	A data.table representing genomic regions.
chr	A string representing the chromosome column.
start	A string representing the name of the start column.
end	A string representing the name of the end column.
strand	A string representing the name of the strand column.
name	A string representing the name of the name column.
splitFactor	A string representing the name of the column to use to split the data.table into multiple data.tables.
metaCols	A string representing the name of the metadata column(s) to include in the returned GRanges object.

Value

A GRanges object.

Examples

```
start1 = c(seq(from=1, to = 2001, by = 1000), 800)
chrString1 = c(rep("chr1", 3), "chr2")
dt = data.table::data.table(chr=chrString1,
                            start=start1,
                            end=start1 + 250)
newGR = dtToGr(dt)
```

---

dtToGrInternal *Two utility functions for converting data.tables into GRanges objects*

Description

Two utility functions for converting data.tables into GRanges objects

Usage

```
dtToGrInternal(DT, chr, start, end = NA, strand = NA, name = NA, metaCols = NA)
```

Arguments

DT	A data.table representing genomic regions.
chr	A string representing the chromosome column.
start	A string representing the name of the start column.
end	A string representing the name of the end column.
strand	A string representing the name of the strand column.
name	A string representing the name of the name column.
metaCols	A string representing the name of the metadata column(s) to include in the returned GRanges object.

Value

A GRanges object.

`exampleOpenSignalMatrix_hg19` *A dataset containing a subset of open chromatin regions across all cell types defined by ENCODE for Homo Sapiens hg19*

Description

Preparation steps:

1. made a universe of regions by merging regions across cell types defined as opened in ENCODE
2. took bigwig files from ENCODE for individual cell types, merged replicates, filtered out blacklisted sites
3. evaluated the signal above regions defined by previous step
4. performed quantile normalization
5. subsetted it

Usage

```
data(exampleOpenSignalMatrix_hg19)
```

Format

data.frame, rows represent whole selection of open chromatin regions across all cell types defined by ENCODE, columns are individual cell types and values are normalized open chromatin signal values.

Source

http://big.databio.org/open_chromatin_matrix/openSignalMatrix_hg19_quantileNormalized_round4.txt.gz

geneModels_hg19 *hg38 gene models*

Description

A dataset containing gene models for Homo Sapiens hg38 genome assembly.

Usage

```
data(geneModels_hg19)
```

Format

A list of two GRanges objects, with genes and exons locations

Source

EnsDb.Hsapiens.v75 package

genomePartitionList *Create a basic genome partition list of genes, exons, introns, UTRs, and intergenic*

Description

Given GRanges for genes, and a GRanges for exons, returns a list of GRanges corresponding to various breakdown of the genome, based on the given annotations; it gives you proximal and core promoters, exons, and introns.

Usage

```
genomePartitionList(
  genesGR,
  exonsGR,
  threeUTRGR = NULL,
  fiveUTRGR = NULL,
  getCorePromoter = TRUE,
  getProxPromoter = TRUE,
  corePromSize = 100,
  proxPromSize = 2000
)
```

Arguments

genesGR	a GRanges object of gene coordinates
exonsGR	a GRanges object of exons coordinates
threeUTRGR	a GRanges object of 3' UTRs
fiveUTRGR	a GRanges object of 5' UTRs
getCorePromoter	option specifying if core promoters should be extracted defeaults to TRUE
getProxPromoter	option specifying if proximal promoters should be extracted defeaults to TRUE
corePromSize	size of core promoter (in bp) upstrem from TSS default value = 100
proxPromSize	size of proximal promoter (in bp) upstrem from TSS default value = 2000

Details

To be used as a partitionList for `calcPartitions`.

Value

A list of GRanges objects, each corresponding to a partition of the genome. Partitions include proximal and core promoters, exons and introns.

Examples

```
partitionList = genomePartitionList(geneModels_hg19$genesGR,
                                    geneModels_hg19$exonsGR,
                                    geneModels_hg19$threeUTRGR,
                                    geneModels_hg19$fiveUTRGR)
```

---

getChromSizes *Returns built-in chrom sizes for a given reference assembly*

Description

Returns built-in chrom sizes for a given reference assembly

Usage

```
getChromSizes(refAssembly)
```

Arguments

refAssembly	A string identifier for the reference assembly

Value

A vector with the chromosome sizes corresponding to a specific genome assembly.

Examples

```
getChromSizes("hg19")
```

---

getChromSizesFromFasta *Get gene models from a remote or local FASTA file*

Description

Get gene models from a remote or local FASTA file

Usage

```
getChromSizesFromFasta(source, destDir = NULL, convertEnsemblUCSC = FALSE)
```

Arguments

source	a string that is either a path to a local or remote FASTA
destDir	a string that indicates the path to the directory where the downloaded FASTA file should be stored
convertEnsemblUCSC	a logical indicating whether Ensembl style chromosome annotation should be changed to UCSC style (add chr)

Value

a named vector of sequence lengths

Examples

```
CElegansFasteCropped = system.file("extdata",
                                   "C_elegans_cropped_example.fa.gz",
                                   package="GenomicDistributions")
CElegansChromSizes = getChromSizesFromFasta(CElegansFasteCropped)
```

getGeneModels *Returns built-in gene models for a given reference assembly*

Description

Some functions require gene models, which can obtained from any source. This function allows you to retrieve a few common built-in ones.

Usage

```
getGeneModels(refAssembly)
```

Arguments

refAssembly	A string identifier for the reference assembly

Value

A list containing the gene models corresponding to a specific reference assembly.

Examples

```
getGeneModels("hg19")
```

getGeneModelsFromGTF *Get gene models from a remote or local GTF file*

Description

Get gene models from a remote or local GTF file

Usage

```
getGeneModelsFromGTF(
  source,
  features,
  convertEnsemblUCSC = FALSE,
  destDir = NULL,
  filterProteinCoding = TRUE
)
```

Arguments

source	a string that is either a path to a local or remote GTF
features	a vector of strings with feature identifiers that to include in the result list
convertEnsemblUCSC	a logical indicating whether Ensembl style chromosome annotation should be changed to UCSC style
destDir	a string that indicates the path to the directory where the downloaded GTF file should be stored
filterProteinCoding	a logical indicating if TSSs should be only protein-coding genes (default = TRUE)

Value

a list of GRanges objects

Examples

```
CElegansGtfCropped = system.file("extdata",
                                 "C_elegans_cropped_example.gtf.gz",
                                 package="GenomicDistributions")
features = c("gene", "exon", "three_prime_utr", "five_prime_utr")
CElegansGeneModels = getGeneModelsFromGTF(CElegansGtfCropped, features, TRUE)
```

---

getGenomeBins *Returns bins used in 'calcChromBins' function Given a named vector of chromosome sizes, the function returns GRangesList object with bins for each chromosome.*

Description

Returns bins used in 'calcChromBins' function Given a named vector of chromosome sizes, the function returns GRangesList object with bins for each chromosome.

Usage

```
getGenomeBins(chromSizes, binCount = 10000)
```

Arguments

chromSizes	a named list of size (length) for each chromosome.
binCount	number of bins (total; *not* per chromosome), defaults to 10,000

Value

A GRangesList object with bins that separate chromosomes into equal parts.

Examples

```
chromSizes = getChromSizes("hg19")
chromBins  = getGenomeBins(chromSizes)
```

`getReferenceData` *Get reference data for a specified assembly*

Description

This is a generic getter function that will return a data object requested, if it is included in the built-in data with the GenomicDistributions package or GenomicDistributionsData package (if installed). Data objects can be requested for different reference assemblies and data types (specified by a tagline, which is a unique string identifying the data type).

Usage

```
getReferenceData(refAssembly, tagline)
```

Arguments

`refAssembly`	Reference assembly string (e.g. 'hg38')
`tagline`	The string that was used to identify data of a given type in the data building step. It's used for the filename so we know what to load, and is what makes this function generic (so it can load different data types).

Value

A requested and included package data object.

`getTssFromGTF` *Get transcription start sites (TSSs) from a remote or local GTF file*

Description

Get transcription start sites (TSSs) from a remote or local GTF file

Usage

```
getTssFromGTF(
  source,
  convertEnsemblUCSC = FALSE,
  destDir = NULL,
  filterProteinCoding = TRUE
)
```

Arguments

source	a string that is either a path to a local or remote GTF
convertEnsemblUCSC	a logical indicating whether Ensembl style chromosome annotation should be changed to UCSC style
destDir	a string that indicates the path to the directory where the downloaded GTF file should be stored
filterProteinCoding	a logical indicating if TSSs should be only protein-coding genes (default = TRUE)

Value

a list of GRanges objects

Examples

```
CElegansGtfCropped = system.file("extdata",
                                 "C_elegans_cropped_example.gtf.gz",
                                 package="GenomicDistributions")
CElegansTss = getTssFromGTF(CElegansGtfCropped, TRUE)
```

---

grToDt *Convert a GenomicRanges into a data.table.*

Description

Convert a GenomicRanges into a data.table.

Usage

```
grToDt(GR)
```

Arguments

GR	A Granges object

Value

A data.table object.

`labelCuts` *Creates labels based on a discretization definition.*

Description

If you are building a histogram of binned values, you want to have labels for your bins that correspond to the ranges you used to bin. This function takes the breakpoints that define your bins and produces nice-looking labels for your histogram plot.

Usage

```
labelCuts(
  breakPoints,
  round_digits = 1,
  signif_digits = 3,
  collapse = "-",
  infBins = FALSE
)
```

Arguments

`breakPoints`	The exact values you want as boundaries for your bins
`round_digits`	Number of digits to cut round labels to.
`signif_digits`	Number of significant digits to specify.
`collapse`	Character to separate the labels
`infBins`	use >/< as labels on the edge bins

Details

`labelCuts` will take a cut group, (e.g., a quantile division of some signal), and give you clean labels (similar to the cut method).

Value

A vector of histogram axis labels.

loadBSgenome *Loads BSgenome objects from UCSC-style character vectors.*

Description

This function will let you use a simple character vector (e.g. 'hg19') to load and then return BSgenome objects. This lets you avoid having to use the more complex annotation for a complete BSgenome object (e.g. BSgenome.Hsapiens.UCSC.hg38.masked)

Usage

```
loadBSgenome(genomeBuild, masked = TRUE)
```

Arguments

genomeBuild	One of 'hg19', 'hg38', 'mm10', 'mm9', or 'grch38'
masked	Should we used the masked version? Default:TRUE

Value

A BSgenome object corresponding to the provided genome build.

Examples

```
## Not run:
bsg = loadBSgenome('hg19')

## End(Not run)
```

loadEnsDb *Load selected EnsDb library*

Description

Load selected EnsDb library

Usage

```
loadEnsDb(genomeBuild)
```

Arguments

genomeBuild	string, genome identifier

Value

loaded library

Examples

```
## Not run:
loadEnsDb("hg19")

## End(Not run)
```

`neighbordt` *Internal helper function to calculate distance between neighboring regions.*

Description

Internal helper function to calculate distance between neighboring regions.

Usage

```
neighbordt(querydt)
```

Arguments

`querydt`	A data table with regions grouped according to chromosome.

Value

A numeric vector with the distances in bp

`nlist` *Nathan's magical named list function. This function is a drop-in replacement for the base list() function, which automatically names your list according to the names of the variables used to construct it. It seamlessly handles lists with some names and others absent, not overwriting specified names while naming any unnamed parameters. Took me awhile to figure this out.*

Description

Nathan's magical named list function. This function is a drop-in replacement for the base list() function, which automatically names your list according to the names of the variables used to construct it. It seamlessly handles lists with some names and others absent, not overwriting specified names while naming any unnamed parameters. Took me awhile to figure this out.

Usage

```
nlist(...)
```

Arguments

...	arguments passed to list()

Value

A named list object.

Examples

```
x=5
y=10
nlist(x,y) # returns list(x=5, y=10)
list(x,y) # returns unnamed list(5, 10)
```

---

plotChromBins *Plot distribution over chromosomes*

Description

Plots result from `genomicDistribution` calculation

Usage

```
plotChromBins(
  genomeAggregate,
  plotTitle = "Distribution over chromosomes",
  ylim = "max"
)
```

Arguments

genomeAggregate	The output from the genomicDistribution function
plotTitle	Title for plot.
ylim	Limit of y-axes. Default "max" sets limit to N of biggest bin.

Value

A ggplot object showing the distribution of the query regions over bins of the reference genome.

Examples

```
agg = data.frame("regionID"=1:5, "chr"=rep(c("chr1"), 5),
                 "withinGroupID"=1:5, "N"=c(1,3,5,7,9))
ChromBins = plotChromBins(agg)
```

plotCumulativePartitions *Plot the cumulative distribution of regions in features*

Description

This function plots the cumulative distribution of regions across a feature set.

Usage

```
plotCumulativePartitions(assignedPartitions, feature_names = NULL)
```

Arguments

assignedPartitions	Results from calcCumulativePartitions
feature_names	An optional character vector of feature names, in the same order as the GenomicRanges or GenomicRangesList object.

Value

A ggplot object of the cumulative distribution of regions in features.

Examples

```
p = calcCumulativePartitionsRef(vistaEnhancers, "hg19")
cumuPlot = plotCumulativePartitions(p)
```

plotDinuclFreq *Plot dinuclotide content within region set(s)*

Description

Given calcDinuclFreq or calcDinuclFreqRef results, this function generates a violin plot of dinucleotide frequency

Usage

```
plotDinuclFreq(DNFDataTable)
```

Arguments

DNFDataTable	A data.table, data.frame, or a list of dinucleotide counts - results from calcDinuclFreq or calcDinuclFreqRef

Value

A ggplot object plotting distribution of dinucleotide content in query regions

Examples

```
DNFDataTable = data.table::data.table(GC = rnorm(400, mean=0.5, sd=0.1),
CG = rnorm(400, mean=0.5, sd=0.5),
AT = rnorm(400, mean=0.5, sd=1),
TA = rnorm(400, mean=0.5, sd=1.5))
DNFPlot =  plotDinuclFreq(DNFDataTable)

## Not run:
query = system.file("extdata", "vistaEnhancers.bed.gz", package="GenomicDistributions")
GRquery = rtracklayer::import(query)
refAssembly = 'hg19'
DNF = calcDinuclFreqRef(GRquery, refAssembly)
DNFPlot2 =  plotDinuclFreq(DNF)

## End(Not run)
```

---

plotExpectedPartitions

Produces a barplot showing how query regions of interest are distributed relative to the expected distribution across a given partition list

---

Description

Produces a barplot showing how query regions of interest are distributed relative to the expected distribution across a given partition list

Usage

```
plotExpectedPartitions(expectedPartitions, feature_names = NULL, pval = FALSE)
```

Arguments

expectedPartitions	A data.frame holding the frequency of assignment to each of the partitions, the expected number of each partition, and the log10 of the observed over expected. Produced by `calcExpectedPartitions`.
feature_names	Character vector with labels for the partitions (optional). By default it will use the names from the first argument.
pval	Logical indicating whether Chi-square p-values should be added for each partition.

Value

A ggplot object using a barplot to show the distribution of the query regions across a given partition list.

Examples

```
p = calcExpectedPartitionsRef(vistaEnhancers, "hg19")
expectedPlot = plotExpectedPartitions(p)
```

`plotFeatureDist` *Plots a histogram of distances to genomic features*

Description

Given the results from `featureDistribution`, plots a histogram of distances surrounding the features of interest

Usage

```
plotFeatureDist(
  dists,
  bgdists = NULL,
  featureName = "features",
  numbers = FALSE,
  nbins = 50,
  size = 1e+05,
  infBins = FALSE,
  tile = FALSE,
  labelOrder = "default"
)
```

Arguments

`dists`	Results from `featureDistribution`
`bgdists`	Background distances. If provided, will plot a background distribution of expected distances
`featureName`	Character vector for plot labels (optional).
`numbers`	a logical indicating whether the raw numbers should be displayed, rather than percentages (optional).
`nbins`	Number of bins on each side of the center point.
`size`	Number of bases to include in plot on each side of the center point.
`infBins`	Include catch-all bins on the sides?
`tile`	Turn on a tile mode, which plots a tiled figure instead of a histogram.
`labelOrder`	– Enter "default" to order by order of user input (default); Enter "center" to order by value in tile in the closest proximity to the center of features (in case TSS is used - center is TSS) (center).

Value

A ggplot2 plot object

Examples

```
TSSdist = calcFeatureDistRefTSS(vistaEnhancers, "hg19")
f = plotFeatureDist(TSSdist, featureName="TSS")
```

`plotGCContent` *Plots a density distribution of GC vectors Give results from the* `calcGCContent` *function, this will produce a density plot*

Description

Plots a density distribution of GC vectors Give results from the `calcGCContent` function, this will produce a density plot

Usage

```
plotGCContent(gcvectors)
```

Arguments

`gcvectors`	A numeric vector or list of numeric vectors of GC contents.

Value

A ggplot object plotting distribution of GC content in query regions.

Examples

```
numVector = rnorm(400, mean=0.5, sd=0.1)
GCplot = plotGCContent(numVector)
vecs = list(example1 = rnorm(400, mean=0.5, sd=0.1),
            example2 = rnorm(600, mean=0.5, sd=0.1))
GCplot = plotGCContent(vecs)
```

`plotNeighborDist` *Plot the distances from regions to their upstream/downstream neighbors or nearest neighbors. Distances can be passed as either raw bp or corrected for the number of regions (log10(obs/exp)), but this has to be specified in the function parameters.*

Description

Plot the distances from regions to their upstream/downstream neighbors or nearest neighbors. Distances can be passed as either raw bp or corrected for the number of regions (log10(obs/exp)), but this has to be specified in the function parameters.

Usage

```
plotNeighborDist(dcvec, correctedDist = FALSE, Nneighbors = FALSE)
```

Arguments

`dcvec`	A numeric vector or list of vectors containing distances to upstream/downstream neighboring regions or to nearest neighbors. Produced by `calcNeighborDist` or `calcNearestNeighbors`
`correctedDist`	A logical indicating if the plot axis should be adjusted to show distances corrected for the number of regions in a regionset.
`Nneighbors`	A logical indicating whether legend should be adjusted if Nearest neighbors are being plotted. Default legend shows distances to upstream/downstream neighbors.

Value

A ggplot density object showing the distribution of raw or corrected distances.

Examples

```
numVector = rnorm(400, mean=5, sd=0.1)
d = plotNeighborDist(numVector)
```

`plotPartitions` *Produces a barplot showing how query regions of interest are distributed across a given partition list*

Description

This function can be used to test a GRanges object against any arbitrary list of genome partitions. The partition list is a priority-ordered list of GRanges objects. Each region in the query will be assigned to a given partition that it overlaps with the highest priority.

Usage

```
plotPartitions(assignedPartitions, numbers = FALSE, stacked = FALSE)
```

Arguments

assignedPartitions	A table holding the frequency of assignment to each of the partitions. Produced by calcPartitions
numbers	logical indicating whether raw overlaps should be plotted instead of the default percentages
stacked	logical indicating that data should be plotted as stacked bar plot

Value

A ggplot object using a barplot to show the distribution of the query regions across a given partition list.

Examples

```
p = calcPartitionsRef(vistaEnhancers, "hg19")
partPlot = plotPartitions(p)
partCounts = plotPartitions(p, numbers=TRUE)
partPlot = plotPartitions(p, stacked=TRUE)
```

---

plotQTHist *Plot quantile-trimmed histogram*

Description

Given the results from calcWidth, plots a histogram with outliers trimmed.

Usage

```
plotQTHist(
  x,
  EndBarColor = "gray57",
  MiddleBarColor = "gray27",
  quantThresh = NULL,
  bins = NULL,
  indep = FALSE,
  numbers = FALSE
)
```

Arguments

x	Data values to plot - vector or list of vectors
EndBarColor	Color for the quantile bars on both ends of the graph (optional)
MiddleBarColor	Color for the bars in the middle of the graph (optional)
quantThresh	Quantile of data to be contained in each end bar (optional) quantThresh values must be under .2, optimal size is under .1
bins	The number of bins for the histogram to allocate data to. (optional)
indep	logical value which returns a list of plots that have had their bins calculated independently; the normal version will plot them on the same x and y axis.
numbers	a logical indicating whether the raw numbers should be displayed, rather than percentages (optional).

Details

x-axis breaks for the frequency calculations are based on the "divisions" results from helper function `calcDivisions`.

Value

A ggplot2 plot object

Examples

```
regWidths = calcWidth(vistaEnhancers)
qtHist = plotQTHist(regWidths)
qtHist2 = plotQTHist(regWidths, quantThresh=0.1)
```

---

plotSummarySignal *The function plotSummarySignal visualizes the signalSummaryMatrix obtained from* `calcSummarySignal`.

Description

The function plotSummarySignal visualizes the signalSummaryMatrix obtained from `calcSummarySignal`.

Usage

```
plotSummarySignal(
  signalSummaryList,
  plotType = "barPlot",
  metadata = NULL,
  colorColumn = NULL,
  filterGroupColumn = NULL,
  filterGroup = NULL
)
```

Arguments

signalSummaryList	Output list from calcSummarySignal function.
plotType	Options are: "jitter" - jitter plot with box plot on top, "boxPlot" - box plot without individual points and outliers, "barPlot" (default) - bar height represents the median signal value for a given cell type, "violinPlot" - violin plot with medians.
metadata	(optional) data.table used for grouping columns from 'signalMatrix' into categories, that are then plotted with different colors. Must contain variable 'colName' that contains all the condition column names from 'signaMatrix'.
colorColumn	(optional only if metadata provided) columns name from 'metadata' table that will be used as grouping variable for coloring.
filterGroupColumn	(optional only if metadata provided and 'filterGroup' specified) allows user to plot specified subgroups only. String specifying the column name in 'metadata' from which groups will be filtered (groups are specified in as 'filterGroups)
filterGroup	(optional only if 'metadata' and 'filterGroupColumn' provided) - string (or vector of strings) of groups from 'filterGroupColumn' to be plottted.

Value

A ggplot object.

Examples

```
signalSummaryList = calcSummarySignal(vistaEnhancers, exampleOpenSignalMatrix_hg19)
metadata = cellTypeMetadata
plotSignal = plotSummarySignal(signalSummaryList)

plotSignalTissueColor = plotSummarySignal(signalSummaryList = signalSummaryList,
plotType = "jitter", metadata = metadata, colorColumn = "tissueType")

plotSignalFiltered = plotSummarySignal(signalSummaryList = signalSummaryList,
plotType = "violinPlot", metadata = metadata, colorColumn = "tissueType",
filterGroupColumn = "tissueType", filterGroup = c("skin", "blood"))
```

---

retrieveFile *Read local or remote file*

Description

Read local or remote file

Usage

```
retrieveFile(source, destDir = NULL)
```

Arguments

source	a string that is either a path to a local or remote GTF
destDir	a string that indicates the path to the directory where the downloaded GTF file should be stored. If not provided, a temporary directory will be used.

Value

data.frame retrieved file path

Examples

```
CElegansGtfCropped = system.file("extdata",
                                 "C_elegans_cropped_example.gtf.gz",
                                 package="GenomicDistributions")
CElegansGtf = retrieveFile(CElegansGtfCropped)
```

---

setB_100 *Example BED file read with rtracklayer::import*

Description

Example BED file read with rtracklayer::import

Usage

```
data(setB_100)
```

Format

GenomicRanges::GRanges

---

splitDataTable *Efficiently split a data.table by a column in the table*

Description

Efficiently split a data.table by a column in the table

Usage

```
splitDataTable(DT, split_factor)
```

Arguments

DT	Data.table to split
split_factor	Column to split, which can be a character vector or an integer.

Value

List of data.table objects, split by column

`theme_blank_facet_label` *Clear ggplot face label.*

Description

Usually ggplot2 facets are labeled with boxes surrounding the label. This function removes the box, so it's a simple label for each facet.

Usage

```
theme_blank_facet_label()
```

Value

A ggplot theme

`TSS_hg19` *hg19 TSS locations*

Description

A dataset containing chromosome sizes for Homo Sapiens hg38 genome assembly

Usage

```
data(TSS_hg19)
```

Format

A named vectors of lengths with one item per chromosome

Source

EnsDb.Hsapiens.v75 package

vistaEnhancers *Example BED file read with rtracklayer::import*

Description

Example BED file read with rtracklayer::import

Usage

```
data(vistaEnhancers)
```

Format

GenomicRanges::GRanges

Index