Package ‘regioneR’

January 31, 2017

Type Package

Title Association analysis of genomic regions based on permutation tests

Version 1.6.2

Date 2016-04-06

Author Anna Diez-Villanueva <adiez@imppc.org>, Roberto Malinverni
       <rmalinverni@carrerasresearch.org> and Bernat Gel <bgel@imppc.org>

Maintainer Bernat Gel <bgel@imppc.org>

Description regioneR offers a statistical framework based on customizable permutation tests to assess the association between genomic region sets and other genomic features.

License Artistic-2.0

Depends memoise, GenomicRanges, BSgenome, rtracklayer, parallel

Imports memoise, GenomicRanges, BSgenome, rtracklayer, parallel, graphics, stats, utils, GenomeInfoDb, IRanges

Suggests BiocStyle, knitr, BSgenome.Hsapiens.UCSC.hg19.masked, testthat

VignetteBuilder knitr

biocViews Genetics, ChIPSeq, DNASeq, MethylSeq, CopyNumberVariation

NeedsCompilation no

RoxygenNote 5.0.1

R topics documented:

characterToBSGenome ........................................... 2
circularRandomizeRegions ..................................... 3
commonRegions .................................................. 4
createFunctionsList ............................................. 5
createRandomRegions ........................................... 6
emptyCacheRegioneR ............................................. 7
extendRegions ................................................... 8
filterChromosomes .............................................. 9
getChromosomesByOrganism ................................... 10
getGenome ....................................................... 10
getGenomeAndMask ............................................. 11
characterToBSGenome

Description

Given a character string with the "name" of a genome, it returns a BSgenome object if available.

Usage

characterToBSGenome(genome.name)

Arguments

genome.name a character string uniquely identifying a BSgenome (e.g. "hg19", "mm10" are ok, but "hg" is not)

Value

A BSgenome object

Note

This function is memoised (cached) using the memoise package. To empty the cache, use forget(characterToBSGenome)
circularRandomizeRegions

See Also
getGenomeAndMask, maskFromBSGenome

Examples

g <- characterToBSGenome("hg19")

circularRandomizeRegions

Circular Randomize Regions

Description
Given a set of regions A and a genome, this function returns a new set of regions created by applying a random spin to each chromosome.

Usage
circularRandomizeRegions(A, genome="hg19", mask=NULL, max.mask.overlap=NULL, max.retries=10, verbose=TRUE, ...)

Arguments
A The set of regions to randomize. A region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)

genome The reference genome to use. A valid genome object. Either a GenomicRanges or data.frame containing one region per whole chromosome or a character uniquely identifying a genome in BSgenome (e.g. "hg19", "mm10" but not "hg"). Internally it uses getGenomeAndMask.

mask The set of regions specifying where a random region can not be (centromeres, repetitive regions, unmappable regions...). A region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, ...). If NULL it will try to derive a mask from the genome (currently only works is the genome is a character string) and if NA it will explicitly give an empty mask.

max.mask.overlap numeric value

max.retries numeric value

verbose a boolean.

... further arguments to be passed to or from methods.

Details
This randomization strategy is useful when the spatial relation between the regions in the RS is important and has to be conserved.

Value
It returns a GenomicRanges object with the regions resulting from the randomization process.
commonRegions

See Also
randomizeRegions, toDataframe, toGRanges, getGenome, getMask, getGenomeAndMask, characterToBSGenome, maskFromBSGenome, resampleRegions, createRandomRegions

Examples
A <- data.frame("chr1", c(1, 10, 20, 30), c(12, 13, 28, 40))
mask <- data.frame("chr1", c(20000000, 100000000), c(22000000, 130000000))
genome <- data.frame(c("chr1", "chr2"), c(1, 1), c(180000000, 200000000))
circularRandomizeRegions(A)
circularRandomizeRegions(A, genome=genome, mask=mask, per.chromosome=TRUE, non.overlapping=TRUE)

commonRegions Common Regions

Description
Returns the regions that are common in two region sets, its intersection.

Usage
commonRegions(A, B)

Arguments
A a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
B a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)

Value
It returns a GenomicRanges object with the regions present in both region sets.

Note
All metadata (additional columns in the region set in addition to chromosome, start and end) will be ignored and not present in the returned region set.

See Also
plotRegions, toDataframe, toGRanges, subtractRegions, splitRegions, extendRegions, joinRegions, mergeRegions, overlapRegions
createFunctionsList

Examples

```r
A <- data.frame("chr1", c(1, 10, 20, 30), c(12, 13, 28, 40))
B <- data.frame("chr1", 25, 35)
commons <- commonRegions(A, B)
plotRegions(list(A, B, commons), chromosome="chr1", regions.labels=c("A", "B", "common"), regions.colors=3:1)
```

---

**createFunctionsList**  
*Create Functions List*

**Description**

Partially applies (the standard Curry function in functional programming) a list of arguments to a function and returns a list of preapplied functions. The result of this function is a list of functions suitable for the multiple evaluation functions in permTest.

**Usage**

```r
createFunctionsList(FUN, param.name, values, func.names)
```

**Arguments**

- **FUN**  
  Function. the function to be partially applied

- **param.name**  
  Character. The name of the parameter to pre-set.

- **values**  
  A list or vector of values to preassign. A function will be created for each of the values in values. If present, the names of the list will be the names of the functions.

- **func.names**  
  Character. The names of the functions created. Useful to identify the functions created. Defaults to the names of the values list or to Function1, Function2... if the values list has no names.

**Value**

It returns a list of functions with parameter param.value pre-set to values.

**Note**

It uses the code posted by "hadley" at http://stackoverflow.com/questions/6547219/how-to-bind-function-arguments

**See Also**

`permTest, overlapPermTest`
Examples

```r
f <- function(a, b) {
  return(a+b)
}
funcs <- createFunctionsList(FUN=f, param.name="b", values=c(1,2,3), func.names=c("plusone", "plustwo", "plusthree"))
funcs$plusone(2)
funcs$plusone(10)
funcs$plusthree(2)
```

```r
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=0, mask=NA)
B <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=0, mask=NA)
overlapsWith <- createFunctionsList(FUN=numOverlaps, param.name="B", values=list(a=A, b=B))
overlapsWith$a(A=A)
overlapsWith$b(A=A)
```

createRandomRegions  Create Random Regions

Description

Creates a set of random regions with a given mean size and standard deviation.

Usage

```r
createRandomRegions(nregions=100, length.mean=250, length.sd=20, genome="hg19", mask=NULL, non.overlapping=TRUE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nregions</td>
<td>The number of regions to be created.</td>
</tr>
<tr>
<td>length.mean</td>
<td>The mean size of the regions created. This is not guaranteed to be the mean of the final region set. See note.</td>
</tr>
<tr>
<td>length.sd</td>
<td>The standard deviation of the region size. This is not guaranteed to be the standard deviation of the final region set. See note.</td>
</tr>
<tr>
<td>genome</td>
<td>The reference genome to use. A valid genome object. Either a GenomicRanges or data.frame containing one region per whole chromosome or a character uniquely identifying a genome in BSgenome (e.g. &quot;hg19&quot;, &quot;mm10&quot; but not &quot;hg&quot;). Internally it uses getGenomeAndMask.</td>
</tr>
<tr>
<td>mask</td>
<td>The set of regions specifying where a random region can not be (centromeres, repetitive regions, unmappable regions...). A region set in any of the accepted formats (GenomicRanges, data.frame, ...). NULL will try to derive a mask from the genome (currently only works is the genome is a character string) and NA explicitly gives an empty mask.</td>
</tr>
<tr>
<td>non.overlapping</td>
<td>A boolean stating whether the random regions can overlap (FALSE) or not (TRUE).</td>
</tr>
</tbody>
</table>
**emptyCacheRegioneR**

**Details**

A set of nregions will be created and randomly placed over the genome. The lengths of the region set will follow a normal distribution with a mean size `length.mean` and a standard deviation `length.sd`. The new regions can be made explicitly non-overlapping by setting `non.overlapping` to TRUE. A mask can be provided so no regions fall in a forbidden part of the genome.

**Value**

It returns a GenomicRanges object with the regions resulting from the randomization process.

**Note**

If the standard deviation of the length is large with respect to the mean, negative lengths might be created. These region lengths will be transformed to into a 1 and so the, for large standard deviations the mean and sd of the lengths are not guaranteed to be the ones in the parameters.

**See Also**

getGenome, getMask, getGenomeAndMask, characterToBSGenome, maskFromBSGenome, randomizeRegions, resampleRegions

**Examples**

```r
gene <- data.frame(c("chr1", "chr2"), c(1, 1), c(180000000, 20000000))
mask <- data.frame("chr1", c(20000000, 100000000), c(22000000, 130000000))
createRandomRegions(nregions=10, length.mean=1000, length.sd=500)
createRandomRegions(nregions=10, genome=genome, mask=mask, non.overlapping=TRUE)
```

---

**emptyCacheRegioneR**

**Empty Cache regioneR**

**Description**

Empties the caches used by the memoised function in the regioneR package.

**Usage**

```r
emptyCacheRegioneR()
```

**Value**

The cache is emptied

**Examples**

```r
emptyCacheRegioneR()
```
extendRegions

Extend Regions

Description

Extends the regions a number of bases at each end. Negative numbers will reduce the region instead of enlarging it.

Usage

extendRegions(A, extend.start=0, extend.end=0)

Arguments

A a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
extend.start an integer. The number of bases to be subtracted from the start of the region.
extend.end an integer. The number of bases to be added at the end of the region.

Value

a GenomicRanges object with the extended regions.

Note

If negative values are provided and the new extremes are "flipped", the function will fail. It does not check if the extended regions fit into the genome.

See Also

plotRegions, toDataFrame, toGRanges, subtractRegions, splitRegions, overlapRegions, commonRegions, mergeRegions, joinRegions

Examples

A <- data.frame("chr1", c(10, 20, 30), c(13, 28, 40))
extend1 <- extendRegions(A, extend.start=5, extend.end=2)
extend2 <- extendRegions(A, extend.start=15)
extend3 <- extendRegions(A, extend.start=-1)
plotRegions(list(A, extend1, extend2, extend3), chromosome="chr1", regions.labels=c("A", "extend1", "extend2", "extend3"))
**Description**

Filters the chromosomes in a region set. It can either filter using a predefined chromosome set (e.g. "autosomal chromosomes in Homo sapiens") or using a custom chromosome set (e.g. only chromosomes "chr22" and "chrX")

**Usage**

```r
filterChromosomes(A, organism="hg", chr.type="canonical", keep.chr=NULL)
```

**Arguments**

- `A`: a region set in any of the formats accepted by `toGRanges` (GenomicRanges, data.frame, etc...)
- `organism`: a character indicating the organism from which to get the predefined chromosome sets. It can be the organism code as used in BSgenome (e.g. hg for human, mm for mouse...) or the full genome assembly identifier, since any digit will be removed to get the organism code.
- `chr.type`: a character indicating the specific chromosome set to be used. Usually "autosomal" or "canonical", although other values could be available for certain organisms.
- `keep.chr`: is a character vector stating the names of the chromosomes to keep. Any chromosome not in the vector will be filtered out. If `keep.chr` is supplied, `organism` and `chr.type` are ignored.

**Value**

A GRanges object containing only the regions in the original region set belonging to the selected chromosomes. All regions in non selected chromosomes are removed.

**See Also**

`getGenomeAndMask`, `listChrTypes`, `getChromosomesByOrganism`

**Examples**

```r
g <- getGenomeAndMask("hg19")$genome
listChrTypes()
g <- filterChromosomes(g, chr.type="autosomal", organism="hg19")
g <- filterChromosomes(g, keep.chr=c("chr1", "chr2", "chr3"))
```
**getChromosomesByOrganism**

**Description**

Function to obtain a list of organisms with their canonical and (when applicable) the autosomal chromosome names. This function is not usually used by the end user directly but through the `filterChromosomes` function.

**Usage**

```
getChromosomesByOrganism()
```

**Value**

a list with the organism as keys and the list of available chromosome sets as values

**See Also**

`getGenome`, `filterChromosomes`

**Examples**

```r
chrsByOrg <- getChromosomesByOrganism()
chrsByOrg["hg"]
chrsByOrg["hg"]["autosomal"]
```

---

**getGenome**

**Description**

Function to obtain a genome

**Usage**

```
getGenome(genome)
```

**Arguments**

- `genome` The genome object or genome identifier.
**getGenomeAndMask**

**Details**

If genome is a **BSgenome** (from the package BioStrings), it will transform it into a **GRanges** with chromosomes and chromosome lengths.

If genome is a **data.frame** with 3 columns, it will transform it into a GRanges.

If genome is a **data.frame** with 2 columns, it will assume the first is the chromosome, the second is the length of the chromosomes and will add 1 as start.

If genome is a character string uniquely identifying a **BSgenome** installed in the system (e.g. "hg19", "mm10"..., but not "hg"), it will create a genome based on the **BSgenome** object identified by the character string.

If genome is a **GRanges** object, it will return it as is.

If genome is non of the above, it will give a warning and try to transform it into a GRanges using **toGRanges**. This can be helpful if genome is a connection to a file.

**Value**

A GRanges object with the "genome" data c(Chromosome, Start (by default, 1), Chromosome Length) given a **BSgenome**, a genome name, a **data.frame** or a GRanges.

A **GRanges** representing the genome with one region per chromosome.

**Note**

This function is memoised (cached) using the **memoise** package. To empty the cache, use `forget(getGenome)`

Please note that passing this function the path to a file will not work, since it will assume the character is the identifier of a genome. To read the genome from a file, please use `getGenome(toGRanges("path/to/file"))`

**See Also**

`getMask`, `getGenomeAndMask`, `characterToBSGenome`, `maskFromBSGenome`, `emptyCacheRegioneR`

**Examples**

```r
getGenome("hg19")
getGenome(data.frame(c("chrA", "chrB"), c(15000000, 10000000)))
```

---

**getGenomeAndMask**

**Description**

Function to obtain a valid genome and mask pair given a valid genome identifier and optionally a mask.

If the genome is not a **BSgenome** object or a character string uniquely identifying a **BSgenome** package installed, it will return the genome "as is". If a mask is provided, it will simply return it. Otherwise it will return the mask returned by `getMask(genome)` or an empty mask if genome is not a valid **BSgenome** or **BSgenome** identifier.
Usage

getGenomeAndMask(genome, mask=NULL)

Arguments

genome the genome object or genome identifier.
mask the mask of the genome in a valid RS format (data.frame, GRanges, BED-like file...). If mask is NULL, it will try to get a mask from the genome. If mask is NA it will return an empty mask. (Default=NULL)

Value

A list with two elements: genome and mask. Genome and mask are GRanges objects.

Note

This function is memoised (cached) using the memoise package. To empty the cache, use forget(getGenomeAndMask)

See Also

getMask, getGenome, characterToBSGenome, maskFromBSGenome, emptyCacheRegioneR

Examples

getGenomeAndMask("hg19", mask=NA)

getGenomeAndMask(genome=data.frame(c("chrA", "chrB"), c(15000000, 1000000)), mask=NA)

getMask

description

Function to obtain a mask given a genome available as a BSgenome. The mask returned is the merge of all the active masks in the BSgenome.

Since it uses characterToBSGenome, the genome can be either a BSgenome object or a character string uniquely identifying the a BSgenome object installed.

Usage

getMask(genome)

Arguments

genome the genome from where the mask will be extracted. It can be either a BSgenome object or a character string uniquely identifying a BSgenome object installed (e.g. "hg19", "mm10", ...)

Value

A GRanges object with the genomic regions to be masked out
Note

This function is memoised (cached) using the `memoise` package. To empty the cache, use `forget(getMask)`

See Also

`getGenome, getGenomeAndMask, characterToBSGenome, maskFromBSGenome, emptyCacheRegioneR`

Examples

```r
hg19.mask <- getMask("hg19")

hg19.mask
```

---

### Description

Joins the regions from a region set A that are less than `min.dist` bases apart.

### Usage

```r
joinRegions(A, min.dist=1)
```

### Arguments

- `A` a region set in any of the accepted formats by `toGRanges` (`GenomicRanges`, `data.frame`, etc...)
- `min.dist` an integer indicating the minimum distance required between two regions in order to not fuse them. Any pair of regions closer than `min.dist` bases will be fused in a larger region. Defaults to 1, so it will only join overlapping regions.

### Value

It returns a `GenomicRanges` object with the regions resulting from the joining process.

### Note

All metadata (additional columns in the region set in addition to chromosome, start and end) will be ignored and not present in the returned region set.

The implementation relies completely in the `reduce` function from `IRanges` package.

### See Also

`plotRegions, toDataframe, toGRanges, subtractRegions, splitRegions, extendRegions, commonRegions, mergeRegions, overlapRegions`
Examples

A <- data.frame("chr1", c(1, 10, 20, 30), c(12, 13, 28, 40))
join1 <- joinRegions(A)
join2 <- joinRegions(A, min.dist=3)
join3 <- joinRegions(A, min.dist=10)
plotRegions(list(A, join1, join2, join3), chromosome="chr1", regions.labels=c("A", "join1", "join2", "join3")

Description

Prints a list of the available organisms and chromosomes sets in the predefined chromosomes sets information.

Usage

listChrTypes()

Value

the list of available chrs and organisms is printed

See Also

filterChromosomes, getChromosomesByOrganism

Examples

g <- getGenomeAndMask("hg19")$genome
listChrTypes()

```R

```
localZScore

**Local z-score**

**Description**
Evaluates the variation of the z-score in the vicinity of the original region set.

**Usage**
localZScore(A, pt, window, step, ...)

**Arguments**
- **A**: a region set in any of the formats accepted by toGRanges (GenomicRanges, data.frame, etc...)
- **pt**: a permTestResult object
- **window**: a window in which the local Z-score will be calculated (bp)
- **step**: the number of bp that divide each Z-score evaluation
- **...**: further arguments to be passed to other methods.

**Value**
It returns a local z-score object.

**See Also**
overlapPermTest, permTest

**Examples**
```r
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FALSE)
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FALSE))
pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE)
plot(pt)
lz <- localZScore(A=A, B=B, pt=pt)
plot(lz)

pt2 <- permTest(A=A, B=B, ntimes=10, randomize.function=randomizeRegions, evaluate.function=list(overlap=numOverlaps, distance=meanDistance), genome=genome, non.overlapping=FALSE)
plot(pt2)
lz2 <- localZScore(A=A, B=B, pt2)
plot(lz2)
```
maskFromBSGenome

Description
Extracts the merge of all the active masks from a BSgenome

Usage
maskFromBSGenome(bsgenome)

Arguments
bsgenome a BSgenome object

Value
A GRanges object with the active mask in the BSgenome

Note
This function is memoised (cached) using the memoise package. To empty the cache, use forget(maskFromBSGenome)

See Also
getGenomeAndMask, characterToBSGenome, emptyCacheRegionR

Examples
```r
g <- characterToBSGenome("hg19")
maskFromBSGenome(g)
```

meanDistance

Mean Distance

Description
Computes the mean distance of regions in A to the nearest element in B

Usage
meanDistance(A, B, ...)

Arguments
A a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
B a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
... any additional parameter needed
**meanInRegions**

**Value**

The mean of the distances of each region in A to the nearest region in B.

**Note**

If a region in A is in a chromosome where no B region is, it will be ignored and removed from the mean computation.

**Examples**

```r
A <- data.frame("chr1", c(1, 10, 20, 30), c(12, 13, 28, 40))
B <- data.frame("chr1", 25, 35)
meanDistance(A, B)
```

---

**Description**

Returns the mean of a value defined by a region set over another set of regions.

**Usage**

```r
meanInRegions(A, x, col.name=NULL, ...)
```

**Arguments**

- **A**: a region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)
- **x**: a region set in any of the accepted formats with an additional column with a value associated to every region. Regions in x can be points (single base regions).
- **col.name**: character indicating the name of the column. If NULL and if a column with the name "value" exist, it will be used. The 4th column will be used otherwise (or the 5th if 4th is the strand).
- **...**: any additional parameter needed

**Value**

It returns a numeric value that is the weighted mean of "value" defined in x over the regions in A. That is, the mean of the value of all regions in x overlapping each region in A weighted according to the number of bases overlapping.

**See Also**

`permTest`
Examples

```r
A <- data.frame("chr1", c(1, 10, 20, 30), c(12, 13, 28, 40))
positions <- sample(1:40, 30)
x <- data.frame("chr1", positions, positions, rnorm(30, 4, 1))
meanInRegions(A, x)
x <- GRanges(seqnames=x[,1], ranges=IRanges(x[,2], end=x[,2]), mcols=x[,3])
meanInRegions(A, x)
```

mergeRegions

Merge Regions

Description

Merges the overlapping regions from two region sets. The two region sets are first merged into one and then overlapping regions are fused.

Usage

```r
mergeRegions(A, B)
```

Arguments

- **A**
  a region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)
- **B**
  a region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)

Value

It returns a GenomicRanges object with the regions resulting from the merging process. Any two overlapping regions from any of the two sets will be fused into one.

Note

All metadata (additional columns in the region set in addition to chromosome, start and end) will be ignored and not present in the returned region set.

The implementation relies completely in the `reduce` function from IRanges package.

See Also

- `plotRegions`
- `toDataframe`
- `toGRanges`
- `subtractRegions`
- `splitRegions`
- `extendRegions`
- `joinRegions`
- `commonRegions`
- `overlapRegions`
Examples

A <- data.frame("chr1", c(1, 5, 20, 30), c(8, 13, 28, 40), x=c(1, 2, 3, 4), y=c("a", "b", "c", "d"))
B <- data.frame("chr1", 25, 35)
merges <- mergeRegions(A, B)
plotRegions(list(A, B, merges), chromosome="chr1", regions.labels=c("A", "B", "merges"), regions.colors=3:1)

<table>
<thead>
<tr>
<th>numOverlaps</th>
<th>Number Of Overlaps</th>
</tr>
</thead>
</table>

Description

Returns the number of regions in A overlapping any region in B

Usage

numOverlaps(A, B, count.once=FALSE, ...)

Arguments

A  a region set in any of the formats accepted by toGRanges (GenomicRanges, data.frame, etc...)
B  a region set in any of the formats accepted by toGRanges (GenomicRanges, data.frame, etc...)
count.once boolean indicating whether the overlap of multiple B regions with a single A region should be counted once or multiple times
... any additional parameters needed

Value

It returns a numeric value that is the number of regions in A overlapping at least one region in B.

See Also

overlapPermTest, permTest

Examples

genie <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALSE)
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FALSE))
numOverlaps(A, B)
numOverlaps(A, B, count.once=TRUE)
overlapGraphicalSummary

Overlap Graphical Summary

Description

Graphical summary of the overlap between two set of regions.

Usage

overlapGraphicalSummary(A, B, regions.labels=c("A","B"), regions.colors=c("black","forestgreen","darkred"), ...)

Arguments

A a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
B a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
regions.labels vector indicating the labels for the y axes.
regions.colors character vector indicating the colors for the regions.
... Arguments to be passed to methods, such as graphical parameters (see par).

@return A plot is created on the current graphics device.

See Also

overlapPermTest, overlapRegions

Examples

A <- data.frame(chr=1, start=c(1,15,24,40,50), end=c(10,20,30,45,55))
B <- data.frame(chr=1, start=c(2,12,28,35), end=c(5,25,33,43))
overlapGraphicalSummary(A, B, regions.labels=c("A","B"), regions.colors=c(4,5,6))

overlapPermTest

Permutation Test for Overlap

Description

Performs a permutation test to see if there is an association in overlap between a region set A and a region set B creating random regions through the genome.

Usage

overlapPermTest (A, B, alternative="auto", ...)

overlapRegions

Arguments

A  a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
B  a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
alternative  the alternative hypothesis must be one of "greater", "less" or "auto". If "auto", the alternative will be decided depending on the data.
...

Value

A list of class permTestResults containing the following components:

• pval the p-value of the test.
• ntimes the number of permutations.
• alternative a character string describing the alternative hypothesis.
• observed the value of the statistic for the original data set.
• permuted the values of the statistic for each permuted data set.
• zscore the value of the standard score. (observed-mean(permuted))/sd(permuted)

See Also

overlapGraphicalSummary, overlapRegions, toDataframe, toGRanges, permTest

Examples

gene <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=1000000, length.sd=20000, genome=genome, non.overlapping=FALSE)
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FALSE))
pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE, verbose=TRUE)
summary(pt)
plot(pt)
plot(pt, plotType="Tailed")

overlapRegions

Overlap Regions

Description

return overlap between 2 regions set A and B

Usage

overlapRegions(A, B, colA=NULL, colB=NULL, type="any", min.bases=1, min.pctA=NULL, min.pctB=NULL,
overlapRegions

Arguments

A a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
B a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)

colA numeric vector indicating which columns of A the results will contain (default NULL)

colB numeric vector indicating which columns of B the results will contain (default NULL)

type
  • AinB: the region in A is contained in a region in B
  • BinA: the region in B is contained in A
  • within: the region in A or B is contained in a region in the other region set
  • equal: the region in A has the same chromosome, start and end as a region in B
  • AleftB: the end of the region from A overlaps the beginning of a region in B
  • ArightB: the start of a region from A overlaps the end of a region in B
  • any: any kind of overlap is returned

min.bases numeric minimum number of bp accepted to define a overlap (default 1)

min.pctA numeric minimum percentage of bases of A accepted to define a overlap (default NULL)

min.pctB numeric minimum percentage of bases of B accepted to define a overlap (default NULL)

get.pctA boolean if TRUE add a column in the results indicating the number percentage of A are involved in the overlap (default FALSE)

get.pctB boolean if TRUE add a column in the results indicating the number percentage of B are involved in the overlap (default FALSE)

get.bases boolean if TRUE add in the results the number of overlapped bases (default FALSE)

only.boolean boolean if TRUE devolve as result a boolean vector containing the overlap state of each regions of A (default FALSE)

only.count boolean if TRUE devolve as result the number of regions of A overlapping with B

... any additional parameter (are there any left?)

Value

the default results is a data.frame with at least 5 columns "chr" indicating the chromosome of the appartenence of each overlap, "startA", "endA", "startB", "endB", indicating the coordinates of the region A and B for each overlap "type" that describe the nature of the overlap (see arguments "type") eventually other columns can be added (see see arguments "colA", "colB", "get.pctA", "get.pctB", "get.bases")

Note

The implementation uses when possible the countOverlaps function from IRanges package.
permTest

See Also
plotRegions, toDataframe, toGRanges, subtractRegions, splitRegions, extendRegions, commonRegions, mergeRegions, joinRegions

Examples
A <- data.frame("chr1", c(1, 5, 20, 30), c(8, 13, 28, 40), x=c(1,2,3,4), y=c("a", "b", "c", "d"))
B <- data.frame("chr1", 25, 35)
overlapRegions(A, B)

permTest  Permutation Test

Description
Performs a permutation test to see if there is an association between a region set and some other feature using an evaluation function.

Usage
permTest(A, ntimes=100, randomize.function, evaluate.function, alternative="auto", min.parallel=1000, force.parallel=NULL, randomize.function.name=NULL, evaluate.function.name=NULL, verbose=FALSE, ...)

Arguments
A  a region set in any of the accepted formats by toGRanges (GenomicRanges, data.frame, etc...)
ntimes  number of permutations
randomize.function  function to create random regions. It must return a set of regions.
evaluate.function  function to search for association. It must return a numeric value.
alternative  the alternative hypothesis must be one of "greater", "less" or "auto". If "auto", the alternative will be decided depending on the data.
min.parallel  if force.parallel is not specified, this will be used to determine the threshold for parallel computation. If length(A) * ntimes > min.parallel, it will activate the parallel computation. Single threaded otherwise.
force.parallel  logical indicating if the computation must be paralleilized.
randomize.function.name  character. If specified, the permTestResults object will have this name instead of the name of the randomization function used. Useful specially when using unnamed anonymous functions.
evaluate.function.name  character. If specified, the permTestResults object will have this name instead of the name of the evaluation function used. Useful specially when using unnamed anonymous functions.
verbose is a boolean. If verbose=TRUE it creates a progress bar to show the computation progress. When combined with parallel computation, it might have an impact in the total computation time.

... further arguments to be passed to other methods.

Details

permTest performs a permutation test of the regions in RS to test the association with the feature evaluated with the evaluation function. The regions are randomized using the randomization function and the evaluation function is used to evaluate them. More information can be found in the vignette.

Value

A list of class permTestResults containing the following components:

- pval the p-value of the test.
- ntimes the number of permutations.
- alternative a character string describing the alternative hypothesis.
- observed the value of the statistic for the original data set.
- permuted the values of the statistic for each permuted data set.
- zscore the value of the standard score. \((\text{observed}-\mu(\text{permuted}))/\sigma(\text{permuted})\)
- randomize.function the randomization function used.
- randomize.function.name the name of the randomization used.
- evaluate.function the evaluation function used.
- evaluate.function.name the name of the evaluation function used.

References


See Also

overlapPermTest

Examples

genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=1000000, length.sd=20000, genome=genome, non.overlapping=FALSE)
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FALSE))
pt2 <- permTest(A=A, B=B, ntimes=10, alternative="auto", verbose=TRUE, genome=genome, evaluate.function=meanDistance, randomize.function=randomizeRegions, non.overlapping=FALSE)
summary(pt2)
plot(pt2)
plot(pt2, plotType="Tailed")
Description

Function for plotting the a `localZScoreResults` object.

Usage

```r
## S3 method for class 'localZScoreResults'
plot(x, main = "", num.x.labels = 5, ...)
```

Arguments

- `x`: an object of class `localZScoreResults`.
- `main`: a character specifying the main title of the plot. Defaults to no title.
- `num.x.labels`: a numeric specifying the number of ticks to label the x axis. The total number will be 2*num.x.labels + 1. Defaults to 5.
- `...`: further arguments to be passed to or from methods.

Value

A plot is created on the current graphics device.

See Also

- `localZScore`

Examples

```r
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALSE)
B <- c(A, createRandomRegions(nregions=10, length.mean=100000, length.sd=20000, genome=genome, non.overlapping=FALSE))
pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE)
lz <- localZScore(A=A, B=B, pt=pt)
plot(lz)
```
Function for plotting the results from a `permTestResults` object.

### Description

Function for plotting the results from a `permTestResults` object.

### Usage

```r
## S3 method for class 'permTestResults'
plot(x, pvalthres = 0.05, plotType = "Tailed",
     main = "", xlab = NULL, ylab = "", ...)
```

### Arguments

- `x`: an object of class `permTestResults`.
- `pvalthres`: p-value threshold for significance. Default is 0.05.
- `plotType`: the type of plot to display. This must be one of "Area" or "Tailed". Default is "Area".
- `main`: a character specifying the title of the plot. Defaults to "".
- `xlab`: a character specifying the label of the x axis. Defaults to NULL, which produces a plot with the evaluation function name as the x axis label.
- `ylab`: a character specifying the label of the y axis. Defaults to "".
- `...`: further arguments to be passed to or from methods.

### Value

A plot is created on the current graphics device.

### See Also

- `permTest`

### Examples

```r
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALSE)
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FALSE))
pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE)
summary(pt)
plot(pt)
plot(pt, plotType="Tailed")

pt2 <- permTest(A=A, B=B, ntimes=10, alternative="auto", genome=genome, evaluate.function=meanDistance, randomize.function=randomizeRegions)
summary(pt2)
plot(pt2)
plot(pt2, plotType="Tailed")
```
Function for plotting the results from a `permTestResultsList` object when more than one evaluation function was used.

### Usage

```r
## S3 method for class 'permTestResultsList'
plot(x, ncol = NA, pvalthres = 0.05,
     plotType = "Tailed", main = "", xlab = NULL, ylab = "", ...)
```

### Arguments

- **x**: an object of class `permTestResultsList`.
- **ncol**: number of plots per row. `ncol=NA` means `ncol=floor(sqrt(length(x)))` so the plot is more or less square (default=NA).
- **pvalthres**: p-value threshold for significance. Default is 0.05.
- **plotType**: the type of plot to display. This must be one of "Area" or "Tailed". Default is "Area".
- **main**: a character specifying the title of the plot. Defaults to "".
- **xlab**: a character specifying the label of the x axis. Defaults to NULL, which produces a plot with the evaluation function name as the x axis label.
- **ylab**: a character specifying the label of the y axis. Defaults to "".
- **...**: further arguments to be passed to or from methods.

### Value

A plot is created on the current graphics device.

### See Also

- `permTest`

### Examples

```r
gene <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALSE)
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FALSE))
pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE)
summary(pt)
plot(pt)
plot(pt, plotType="Tailed")
```
plotRegions

Description

Plots sets of regions

Usage

plotRegions(x, chromosome, start=NULL, end=NULL, regions.labels=NULL, regions.colors=NULL, ...)

Arguments

x list of objects to be plotted.
chromosome character or numeric value indicating which chromosome you want to plot.
start numeric value indicating from which position you want to plot.
end numeric value indicating to which position you want to plot.
regions.labels vector indicating the labels for the y axes. It must have the same length as x.
regions.colors character vector indicating the colors for the plotted regions. It must have the same length as x.
... Arguments to be passed to methods, such as graphical parameters (see `par`).

Value

A plot is created on the current graphics device.

Examples

A <- data.frame(chr=1, start=c(1,15,24,40,50), end=c(10,20,30,45,55))
B <- data.frame(chr=1, start=c(2,12,28,35), end=c(5,25,33,43))
plotRegions(list(A,B), chromosome=1, regions.labels=c("A","B"), regions.colors=3:2)
randomizeRegions  

**Description**

Given a set of regions A and a genome, this function returns a new set of regions randomly distributed in the genome.

**Usage**

```r
randomizeRegions(A, genome="hg19", mask=NULL, allow.overlaps=TRUE, per.chromosome=FALSE, ...)
```

**Arguments**

- `A` The set of regions to randomize. A region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)
- `genome` The reference genome to use. A valid genome object. Either a GenomicRanges or data.frame containing one region per whole chromosome or a character uniquely identifying a genome in BSgenome (e.g. "hg19", "mm10"... but not "hg"). Internally it uses `getGenomeAndMask`.
- `mask` The set of regions specifying where a random region can not be (centromeres, repetitive regions, unmappable regions...). A region set in any of the accepted formats by `toGRanges` (GenomicRanges,data.frame, ...). If `NULL` it will try to derive a mask from the genome (currently only works if the genome is a character string). If `NA` it gives, explicitly, an empty mask.
- `allow.overlaps` A boolean stating whether the random regions can overlap (FALSE) or not (TRUE).
- `per.chromosome` Boolean. If TRUE, the regions will be created in a per chromosome manner - every region in A will be moved into a random position at the same chromosome where it was originally.
- `...` further arguments to be passed to or from methods.

**Details**

The new set of regions will be created with the same sizes of the original ones, and optionally placed in the same chromosomes.

In addition, they can be made explicitly non overlapping and a mask can be provided so no regions fall in an undesirable part of the genome.

**Value**

It returns a GenomicRanges object with the regions resulting from the randomization process.

**See Also**

toDataframe, toGRanges, getGenome, getMask, getGenomeAndMask, characterToBSGenome, maskFromBSGenome, resampleRegions, createRandomRegions, circularRandomizeRegions
Examples

```r
A <- data.frame("chr1", c(1, 10, 20, 30), c(12, 13, 28, 40))
mask <- data.frame("chr1", c(20000000, 100000000), c(22000000, 130000000))
genome <- data.frame(c("chr1", "chr2"), c(1, 1), c(180000000, 20000000))
randomizeRegions(A)
randomizeRegions(A, genome=genome, mask=mask, per.chromosome=TRUE, allow.overlaps=FALSE)
```

---

recomputePermTest  
Recompute Permutation Test

Description

Recomputes the permutation test changing the alternative hypotesis

Usage

```r
recomputePermTest(ptr)
```

Arguments

- `ptr`  
an object of class `permTestResults`

Value

A list of class `permTestResults` containing the same components as `permTest` results.

See Also

- `permTest`

Examples

```r
A <- createRandomRegions(nregions=10, length.mean=1000000)
B <- createRandomRegions(nregions=10, length.mean=1000000)
resPerm <- permTest(A=A, B=B, ntimes=5, alternative="less", genome="hg19", evaluate.function=meanDistance, randomize.function=randomizeRegions)
plot(resPerm)
```
resampleRegions  

**Resample Regions**

**Description**

Function for sampling a region set from a universe of region sets.

**Usage**

```r
resampleRegions(A, universe, per.chromosome=FALSE, ...)
```

**Arguments**

- `A`  a region set in any of the formats accepted by `toGRanges` (GenomicRanges, data.frame, etc...)
- `universe`  a region set in any of the formats accepted by `toGRanges` (GenomicRanges, data.frame, etc...)
- `per.chromosome`  boolean indicating if sample must be by chromosome.
- `...`  further arguments to be passed to or from methods.

**Value**

a GenomicRanges object. A sample from the universe with the same length as A.

**See Also**

toDataframe, toGRanges, randomizeRegions, createRandomRegions

**Examples**

```r
universe <- data.frame(chr=1, start=c(1,15,24,40,50), end=c(10,20,30,45,55))
A <- data.frame(chr=1, start=c(2,12,28,35), end=c(5,25,33,43))
resampleRegions(A, universe, per.chromosome=TRUE)
```

splitRegions  

**Split Regions**

**Description**

Splits a region set A by both ends of the regions in a second region set B.

**Usage**

```r
splitRegions(A, B, min.size=1, track.original=TRUE)
```
subtractRegions

**Description**

Function for subtracting a region set from another region set.

**Usage**

```r
subtractRegions(A, B)
```

**Arguments**

- **A**: a region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)
- **B**: a region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)

**Details**

This function returns the regions in A minus the parts of them overlapping the regions in B. Overlapping regions in the result will be fused.

The implementation relies completely in the `setdiff` function from IRanges package.

**Examples**

```r
A <- data.frame(chr=1, start=c(1, 15, 24, 40, 50), end=c(10, 20, 30, 45, 55))
B <- data.frame(chr=1, start=c(2, 12, 28, 35), end=c(5, 25, 33, 43))
splits <- splitRegions(A, B)
plotRegions(list(A, B, splits), chromosome=1, regions.labels=c("A", "B", "splits"), regions.colors=3:1)
```
toDataframe

Value
A GenomicRanges object

Examples
A <- data.frame(chr=1, start=c(1, 15, 24, 31), end=c(10, 20, 30, 35))
B <- data.frame(chr=1, start=c(2, 12, 24, 35), end=c(5, 25, 29, 40))
subtract <- subtractRegions(A, B)
plotRegions(list(A, B, subtract), chromosome=1, regions.labels=c("A", "B", "subtract"), regions.colors=3:1)

Description
Transforms a GRanges object or a data.frame containing a region set into a data.frame.

Usage
toDataframe(A, stranded=FALSE)

Arguments
A a GRanges object.
stranded (only used when A is a GRanges object) a logical indicating whether a column with the strand information have to be added to the result (Defaults to FALSE)

Details
If the object is of class data.frame, it will be returned untouched.

Value
A data.frame with the regions in A. If A was a GRanges object, the output will include any metadata present in A.

See Also
toGRanges

Examples
A <- data.frame(chr=1, start=c(1, 15, 24), end=c(10, 20, 30), x=c(1,2,3), y=c("a", "b", "c"))
A2 <- toGRanges(A)
toDataframe(A2)
Description

Transforms a file or an object containing a region set into a \texttt{GRanges} object.

Usage

toGRanges(A, ...)

Arguments

A

a \texttt{data.frame} containing a region set, a \texttt{GRanges} object, a BED file or any type
of file supported by \texttt{rtracklayer}

... further arguments to be passed to other methods.

Details

If A is already a \texttt{GRanges} object, it will be returned untouched.

If A is a file name or connection to a file in any of the formats supported by \texttt{rtracklayer}'s import
function (BED, GFF...) it will be imported using \texttt{rtracklayer}.

If A is a data frame, the function will assume the first three columns are chromosome, start and end
and create a \texttt{GRanges} object. Any additional column will be considered metadata and stored as such
in the \texttt{GRanges} object.

Value

A \texttt{GRanges} object with the regions in A

See Also

toDataframe

Examples

A <- data.frame(chr=1, start=c(1, 15, 24), end=c(10, 20, 30), x=c(1,2,3), y=c("a", "b", "c"))
toGRanges(A)
**uniqueRegions**

<table>
<thead>
<tr>
<th>uniqueRegions</th>
<th>Unique Regions</th>
</tr>
</thead>
</table>

**Description**

Returns the regions unique to only one of the two region sets, that is, all parts of the genome covered by only one of the two region sets.

**Usage**

```
uniqueRegions(A, B)
```

**Arguments**

- `A`  
  a region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)

- `B`  
  a region set in any of the accepted formats by `toGRanges` (GenomicRanges, data.frame, etc...)

**Value**

It returns a GenomicRanges object with the regions unique to one of the region sets.

**Note**

All metadata (additional columns in the region set in addition to chromosome, start and end) will be ignored and not present in the returned region set.

**See Also**

`toGRanges, subtractRegions, commonRegions, mergeRegions`

**Examples**

```
A <- data.frame("chr1", c(1, 10, 20, 30), c(12, 13, 28, 40))
B <- data.frame("chr1", 25, 35)
uniques <- uniqueRegions(A, B)
plotRegions(list(A, B, uniques), chromosome="chr1", regions.labels=c("A", "B", "uniques"), regions.colors=3:1)
```
Index

BSgenome, 2, 3, 6, 9, 11, 12, 16, 29
characterToBSGenome, 2, 4, 7, 11–13, 16, 29
circularRandomizeRegions, 3, 29
countOverlaps, 22
createFunctionsList, 5
createRandomRegions, 4, 6, 29, 31
data.frame, 3, 4, 6, 8, 9, 11, 13, 15–23, 29, 31–35
defactor, 3, 4, 6, 8, 9, 11, 13, 15–23, 29, 31–35
default, 30
extendRegions, 4, 8, 13, 18, 23, 32
filterChromosomes, 9, 10, 14
forget, 11–13, 16
GenomicRanges, 3, 4, 6–9, 13, 15–23, 29, 31, 32, 35
getChromosomesByOrganism, 9, 10, 14
getGenome, 4, 7, 10, 12, 13, 29
getGenomeAndMask, 3, 4, 6, 7, 9, 11, 13, 16, 29
getMask, 4, 7, 11, 12, 12, 29
GRanges, 9, 11, 12, 16, 33, 34
joinRegions, 4, 8, 13, 18, 23, 32
listChrTypes, 9, 14
localZScore, 15, 25
maskFromBSGenome, 3, 4, 7, 11–13, 16, 29
mean, 21, 24
meanDistance, 16
memoise, 11–13, 16
mergeRegions, 4, 8, 13, 18, 23, 32, 35
NA, 3, 6, 12, 29
NULL, 3, 6, 12, 29
numOverlaps, 19
overlapGraphicalSummary, 20, 21
overlapPermTest, 5, 15, 19, 20, 20, 24
par, 20, 28
permTest, 5, 15, 17, 19, 21, 23, 26, 27, 30
plot.localZScoreResults, 25
plot.permTestResults, 26
plot.permTestResultsList, 27
plotRegions, 4, 8, 13, 18, 23, 28
randomizeRegions, 4, 7, 29, 31
recomputePermTest, 30
reduce, 13, 18
resampleRegions, 4, 7, 29, 31
sd, 21, 24
splitRegions, 4, 8, 13, 18, 23, 31
subtractRegions, 4, 8, 13, 18, 23, 32, 32, 35
toDataframe, 4, 8, 13, 18, 21, 23, 29, 31, 32, 33, 34
toGRanges, 3, 4, 8, 9, 11, 13, 15–23, 29, 31–33, 34, 35
uniqueRegions, 35