Package 'regioneR'

November 12, 2026

Type Package

Title Association analysis of genomic regions based on permutation tests

Version 1.44.0

Date 2025-06-20

Author Anna Diez-Villanueva <adiez@iconcologia.net>, Roberto Malinverni <roberto.malinverni@gmail.com> and Bernat Gel <bgel@igtp.cat>

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 GenomicRanges

Imports memoise, GenomicRanges, IRanges, BSgenome, Biostrings, rtracklayer, parallel, graphics, stats, utils, methods, Seqinfo, GenomeInfoDb, S4Vectors, tools

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

VignetteBuilder knitr

Encoding UTF-8

biocViews Genetics, ChIPSeq, DNASeq, MethylSeq, CopyNumberVariation

NeedsCompilation no

RoxygenNote 7.3.2

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

git_branch RELEASE_3_23

git_last_commit 550014a

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

characterToBSGenome . . . 3
circularRandomizeRegions . . . 3
commonRegions . . . 5
createFunctionsList . . . 6
createRandomRegions . . . 7
emptyCacheRegioneR . . . 8
extendRegions . . . 9
filterChromosomes . . . 10
getChromosomesByOrganism . . . 11
getGenome . . . 11
getGenomeAndMask . . . 12
getMask . . . 13
joinRegions . . . 14
listChrTypes . . . 15
localZScore . . . 16
maskFromBSGenome . . . 17
meanDistance . . . 17
meanInRegions . . . 18
mergeRegions . . . 19
numOverlaps . . . 20
overlapGraphicalSummary . . . 21
overlapPermTest . . . 22
overlapRegions . . . 23
permTest . . . 25
plot.localZScoreResults . . . 26
plot.localZScoreResultsList . . . 27
plot.permTestResults . . . 28
plot.permTestResultsList . . . 30
plotRegions . . . 31
print.permTestResults . . . 32
randomizeRegions . . . 32
recomputePermTest . . . 34
resampleGenome . . . 35
resampleRegions . . . 36
splitRegions . . . 37
subtractRegions . . . 38
summary.permTestResults . . . 39
summary.permTestResultsList . . . 39
toDataframe . . . 40
toGRanges . . . 41
uniqueRegions . . . 43

Index **45**

characterToBSGenome *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)

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=
```

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.

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, 20000000))

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

Examples

```
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

```
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

```
f <- function(a, b) {
 return(a+b)
}

funcs <- createFunctionsList(FUN=f, param.name="b", values=c(1,2,3), func.names=c("plusone", "plustwo", "plusthre

funcs$plusone(2)
funcs$plusone(10)
funcs$plusthree(2)
```

```
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

```
createRandomRegions(nregions=100, length.mean=250, length.sd=20, genome="hg19", mask=NULL, non.overla
```

Arguments

`nregions`	The number of regions to be created.
`length.mean`	The mean size of the regions created. This is not guaranteed to be the mean of the final region set. See note.
`length.sd`	The standard deviation of the region size. This is not guaranteed to be the standard deviation of the final region set. See note.
`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 (`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.
`non.overlapping`	A boolean stating whether the random regions can overlap (FALSE) or not (TRUE).

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 transfromed 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

```
genome <- 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 functions in the regioneR package.

Usage

```
emptyCacheRegioneR()
```

Value

The cache is emptied

Examples

```
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", "e
```

filterChromosomes *filterChromosomes*

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

```
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", althought 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

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

getChromosomesByOrganism *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

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

getGenome *getGenome*

Description

Function to obtain a genome

Usage

```
getGenome(genome)
```

Arguments

genome	The genome object or genome identifier.

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

```
getGenome("hg19")

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

---

getGenomeAndMask *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, 10000000)), mask=NA)
```

---

getMask *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

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

hg19.mask
```

---

joinRegions *Join Regions*

Description

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

Usage

```
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"), re
```

---

listChrTypes *filterChromosomes listChrTypes*

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()

g <- filterChromosomes(g, chr.type="autosomal", organism="hg19")
```

localZScore *Local z-score*

Description

Evaluates tthe variation of the z-score in the vicinty 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 wich 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

```
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=FAL

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=numOver
plot(pt2)

lz2 <- localZScore(A=A, B=B, pt2)
plot(lz2)
```

maskFromBSGenome *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, emptyCacheRegioneR

Examples

```
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

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

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

B <- data.frame("chr1", 25, 35)

meanDistance(A, B)
```

---

meanInRegions *Mean In Regions*

Description

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

Usage

```
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

```
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

```
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)
```

---

numOverlaps *Number Of Overlaps*

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

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FAL

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","dar
```

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 the overlap between two sets of regions A and B is higher (or lower) than expected by chance. It will internally call `permTest` with the appropiate parameters to perform the permutation test. If B is a list or a GRangesList, it will perform one permutation test per element of the list, testing the overlap between A and each element of B independently.

Usage

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

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

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 hypotesis.
- `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)`

Note

IMPORTANT: Since it uses `link{permTest}` internally, it is possible to use most of the parameters of that function in `overlapPermTest`, including: `ntimes`, `force.parallel`, `min.parallel` and `verbose`. In addition, this function accepts most parameters of the `randomizeRegions` function including `genome`, `mask`, `allow.overlaps` and `per.chromosome` and the parameters of `numOverlaps` such as `count.once`.

See Also

`overlapGraphicalSummary`, `overlapRegions`, `toDataframe`, `toGRanges`, `permTest`

Examples

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FAL

pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE, verbose=TRUE)
summary(pt)
plot(pt)
plot(pt, plotType="Tailed")


C <- c(B, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FAL
pt <- overlapPermTest(A=A, B=list(B=B, C=C), ntimes=10, genome=genome, non.overlapping=FALSE, verbose=TRUE)
summary(pt)
plot(pt)
```

`overlapRegions` *Overlap Regions*

Description

return overlap between 2 regios set A and B

Usage

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

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 minimun number of bp accepted to define a overlap (default 1)
min.pctA	numeric minimun percentage of bases of A accepted to define a overlap (default NULL)
min.pctB	numeric minimun 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.

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,
```

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 paralelized.
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	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 hypotesis.
- `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)`
- `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

Davison, A. C. and Hinkley, D. V. (1997) Bootstrap methods and their application, Cambridge University Press, United Kingdom, 156-160

See Also

`overlapPermTest`

Examples

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FAL


pt2 <- permTest(A=A, B=B, ntimes=10, alternative="auto", verbose=TRUE, genome=genome, evaluate.function=meanDista
summary(pt2)
plot(pt2)
plot(pt2, plotType="Tailed")
```

---

plot.localZScoreResults

Plot localZscore results

Description

Function for plotting the a `localZScoreResults` object.

Usage

```
## 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

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=100000, length.sd=20000, genome=genome, non.overlapping=FA

pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE)

lz <- localZScore(A=A, B=B, pt=pt)
plot(lz)
```

---

plot.localZScoreResultsList

Plot a list of localZscore results

Description

Function for plotting the a localZScoreResultsList object.

Usage

```
## S3 method for class 'localZScoreResultsList'
plot(x, ncol = NA, main = "", num.x.labels = 5, ...)
```

Arguments

x	an object of class localZScoreResultsList.
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

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=100000, length.sd=20000, genome=genome, non.overlapping=FA

pt <- overlapPermTest(A=A, B=B, ntimes=10, genome=genome, non.overlapping=FALSE)

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=numOver
plot(pt2)
```

---

plot.permTestResults *Function for plotting the results from a* permTestResults *object.*

Description

Function for plotting the results from a permTestResults object.

Usage

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

```
  ylab = "",
  ylim = NULL,
  xlim = NULL,
  ...
)
```

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 "".
ylim	defines the y limits of the plot. Passed to the underlying plot call.
xlim	defines the x limits of the plot. Passed to the underlying plot call.
...	further arguments to be passed to or from methods.

Value

A plot is created on the current graphics device.

See Also

permTest

Examples

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FA

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
summary(pt2)
plot(pt2)
plot(pt2, plotType="Tailed")
```

`plot.permTestResultsList` *Function for plotting the results from a* `permTestResultsList` *object when more than one evaluation function was used.*

Description

Function for plotting the results from a `permTestResultsList` object when more than one evaluation function was used.

Usage

```
## 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

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FA

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=list(distance=meanDista
summary(pt2)
plot(pt2)
plot(pt2, plotType="Tailed")
```

---

plotRegions *Plot Regions*

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 ploted.
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)
```

---

print.permTestResults *Print permTestResults objects*

Description

Print permTestResults objects

Usage

```
## S3 method for class 'permTestResults'
print(x, ...)
```

Value

the object is printed

Examples

```
genome <- filterChromosomes(getGenome("hg19"), keep.chr="chr1")
A <- createRandomRegions(nregions=20, length.mean=10000000, length.sd=20000, genome=genome, non.overlapping=FALS
B <- c(A, createRandomRegions(nregions=10, length.mean=10000, length.sd=20000, genome=genome, non.overlapping=FAL

pt <- permTest(A=A, B=B, ntimes=10, alternative="auto", verbose=TRUE, genome=genome, evaluate.function=meanDistan
print(pt)
```

---

randomizeRegions *Randomize Regions*

Description

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

Usage

```
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 maner - 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.

Note

randomizeRegions assumes that chromosomes start at base 1. If a chromosome starts at another base number, for example at base 1000, random regions might appear in the [1:1000] interval. This should not affect most uses of randomizeRegions, but might be important in some advanced analysis involving artificially contructed genomes.

See Also

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

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, 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

```
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

```
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, random

plot(resPerm)
```

resampleGenome *resampleGenome*

Description

Fast alternative to randomizeRegions. It creates a tiling (binning) of the whole genome with tiles the mean size of the regions in A and then places the regions by sampling a length(A) number of tiles and placing the resampled regions there.

Usage

```
resampleGenome(A, simple = FALSE, per.chromosome = FALSE, genome="hg19", min.tile.width=1000, ...)
```

Arguments

A	an object of class GenomigRanges
simple	logical, if TRUE the randomization process will not take into account the specific width of each region in A. (defalut = FALSE)
per.chromosome	logical, if TRUE the randomization will be perform by chromosome. (default = TRUE)
genome	character or GenomicRanges, genome using for the randomization
min.tile.width	integer, the minimum size of the genome tiles. If they are too small, the functions gets very slow and may even fail to work. (default = 1000, 1kb tiles)
...	further arguments to be passed to other methods.

Value

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

See Also

toDataframe, toGRanges, randomizeRegions, createRandomRegions

Examples

```
A <- data.frame(chr=1, start=c(2,12,28,35), end=c(5,25,33,43))

B <- resampleGenome(A)
B
width(B)

B2 <- resampleGenome(A, simple=TRUE)
B2
width(B2)

resampleGenome(A, per.chromosome=TRUE)
```

resampleRegions *Resample Regions*

Description

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

Usage

```
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 univers with the same length as A.

See Also

toDataframe, toGRanges, randomizeRegions, createRandomRegions

Examples

```
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

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

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...)
min.size	numeric value, minimal size of the new regions
track.original	logical indicating if you want to keep the original regions and additional information in the output

Value

A GRanges with the splitted regions.

See Also

toDataframe, toGRanges, subtractRegions, commonRegions, extendRegions, joinRegions, mergeRegions, 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))

splits <- splitRegions(A, B)

plotRegions(list(A, B, splits), chromosome=1, regions.labels=c("A", "B", "splits"), regions.colors=3:1)
```

subtractRegions *Subtract Regions*

Description

Function for subtracting a region set from another region set.

Usage

```
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.

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)
```

summary.permTestResults *Summary of permTestResults objects*

Description

Summary of permTestResults objects

Usage

```
## S3 method for class 'permTestResults'
summary(object, ...)
```

Value

the summary is printed

summary.permTestResultsList *Summary of permTestResultsList objects*

Description

Summary of permTestResultsList objects

Usage

```
## S3 method for class 'permTestResultsList'
summary(object, ...)
```

Value

the summary is printed

toDataframe *toDataframe*

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 oject 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)
```

toGRanges *toGRanges*

Description

Transforms a file or an object containing a region set into a GRanges object.

Usage

```
toGRanges(A, ..., genome=NULL, sep=NULL, comment.char="#")
```

Arguments

A	a data.frame containing a region set, a GRanges object, a BED file, any type of file supported by rtracklayer::import or a "SimpleRleList" returned by GenomicRanges::coverage. If there are more than 1 argument, it will build a dataframe out ouf them and process it as usual. If there's only a single argument and it's a character, if it's not an existing file name it will be treated as the definition of a genomic region in the UCSC/IGV format (i.e. "chr9:34229289-34982376") and parsed.
...	further arguments to be passed to other methods.
genome	(character or BSgenome) The genome info to be attached to the created GRanges. If NULL no genome info will be attached. (defaults to NULL)
sep	(character) The field separator in the text file. If NULL it will be automatically guessed. Only used when reading some file formats. (Defaults to NULL)
comment.char	(character) The character marking comment lines. Only used when reading some file formats. (Defaults to "#")

Details

If A is already a GRanges object, it will be returned untouched.

If A is a data frame, the function will assume the first three columns are chromosome, start and end and create a GRanges object. Any additional column will be considered metadata and stored as such in the GRanges object. There are 2 special cases: 1) if A is a data.frame with only 2 columns, it will assume the first one is the chromosome and the second one the position and it will create a GRanges with single base regions and 2) if the data.frame has the first 3 columns named "SNP", "CHR" and "BP" it will shuffle the columns and repeat "BP" to build a GRanges of single base regions (this is the standard ouput format of plink).

If A is not a data.frame and there are more parameters, it will try to build a data.frame with all parameters and use that data.frame to build the GRanges. This allows the user to call it like `toGRanges("chr1", 10, 20)`.

If A is a character or a character vector and it's not a file or a URL, it assumes it's a genomic position description in the form used by UCSC or IGV, "chr2:1000-2000". It will try to parse the character strings into chromosome, start and end and create a GRanges. The parser can deal with commas

separating thousands (e.g. "chr2:1,000-2,000") and with the comma used as a start/end separator (e.g. "chr2:1000,2000"). These different variants can be mixed in the same character vector.

If A is a "SimpleRleList" it will be interpreted as the result from GenomicRanges::coverage and the function will return a GRanges with a single metadata column named "coverage".

If A is a file name (local or remote) or a connection to a file, it will try to load it in different ways: * BED files (identified by a "bed" extension): will be loaded using `rtracklayer::import` function. Coordinates are 0 based as described in the BED specification (https://genome.ucsc.edu/FAQ/FAQformat.html#format1). * PLINK assoc files (identified by ".assoc", ".assoc.fisher", ".assoc.dosage", ".assoc.linear", ".assoc.logistic"): will be loaded as single-base ranges with all original columns present and the SNPs ids as the ranges names * Any other file: It assumes the file is a "generic" tabular file. To load it it will ignore any header line starting with `comment.char`, autodetect the field separator (if not provided by the user), autodetect if it has a header and read it accordingly.

The genome parameter can be used to set the genome information of the created GRanges. It can be either a BSgenome object or a character string defining a genome (e.g. "hg19", "mm10"...) as accepted by the `BSgenome::getBSgenome` function. If a valid genome is given and the corresponding BSgenome package is installed, the genome information will be attached to the GRanges. If the chromosome naming style from the GRanges and the genome object are different, it will try to change the GRanges styles to match those of the genome using `GenomeInfoDb::seqlevelsStyle`.

Value

A GRanges object with the regions in A

Note

IMPORTANT: Regarding the coordinates, BED files are 0 based while `data.frames` and generic files are treated as 1 based. Therefore reading a line "chr9 100 200" from a BED file will create a 99 bases wide interval starting at base 101 and ending at 200 but reading it from a txt file or from a `data.frame` will create a 100 bases wide interval starting at 100 and ending at 200. This is specially relevant in 1bp intervals. For example, the 10th base of chromosome 1 would be "chr1 9 10" in a BED file and "chr1 10 10" in a txt file.

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"))
gr1 <- toGRanges(A)

#No need to give the data.frame columns any specific name
A <- data.frame(1, c(1, 15, 24), c(10, 20, 30),  x=c(1,2,3), y=c("a", "b", "c"))
gr2 <- toGRanges(A)

#We can pass the data without building the data.frame
gr3 <- toGRanges("chr9", 34229289, 34982376, x="X")

#And each argument can be a vector (they will be recycled as needed)
```

```
gr4 <- toGRanges("chr9", c(34229289, 40000000), c(34982376, 50000000), x="X", y=c("a", "b"))

#toGRanges will automatically convert the second and third argument into numerics
gr5 <- toGRanges("chr9", "34229289", "34982376")

#It can be a file from disk
bed.file <- system.file("extdata", "my.special.genes.txt", package="regioneR")
gr6 <- toGRanges(bed.file)

#Or a URL to a valid file
#gr7 <- toGRanges("http://path.to/myfile.bed")

#It can also parse genomic location strings
gr8 <- toGRanges("chr9:34229289-34982376")

#more than one
gr9 <- toGRanges(c("chr9:34229289-34982376", "chr10:1000-2000"))

#even with strange and mixed syntaxes
gr10 <- toGRanges(c("chr4:3873-92928", "chr4:3873,92928", "chr5:33,444-45,555"))

#if the genome is given it is used to annotate the resulting GRanges
gr11 <- toGRanges(c("chr9:34229289-34982376", "chr10:1000-2000"), genome="hg19")


#and the genome is added to the GRanges even if A is a GRanges
gr12 <- toGRanges(gr6, genome="hg19")

#And it will change the chromosome naming of the GRanges to match that of the
#genome if it is possible (using GenomeInfoDb::seqlevelsStyle)
gr2
gr13 <- toGRanges(gr2, genome="hg19")

#in addition, it can convert other objects into GRanges such as the
#result of GenomicRanges::coverage

gr14 <- toGRanges(c("1:1-20", "1:5-25", "1:18-40"))
cover <- GenomicRanges::coverage(gr14)
gr15 <- toGRanges(cover)
```

---

uniqueRegions *Unique Regions*

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

∗ **internal**
    plot.localZScoreResultsList, 27
    print.permTestResults, 32
    summary.permTestResults, 39
    summary.permTestResultsList, 39

BSgenome, *3, 4*, *7*, *10*, *12, 13*, *17*, *33*, *42*

characterToBSGenome, 3, *4*, *8*, *12–14*, *17*, *33*
circularRandomizeRegions, 3, *33*
commonRegions, 5, *9*, *15*, *20*, *24*, *37*, *44*
countOverlaps, *24*
createFunctionsList, 6
createRandomRegions, *4*, 7, *33*, *35, 36*

data.frame, *4, 5*, *7*, *9, 10*, *12*, *14*, *16*, *18–25*, *33*, *36–38*, *40, 41*, *44*

emptyCacheRegioneR, 8, *12–14*, *17*
extendRegions, *5*, 9, *15*, *20*, *24*, *37*

filterChromosomes, 10, *11*, *15*
forget, *12–14*, *17*

GenomicRanges, *4, 5*, *7–10*, *14*, *16*, *18–23*, *25*, *33*, *35–38*, *44*
getChromosomesByOrganism, *10*, 11, *15*
getGenome, *4*, *8*, *11*, 11, *13, 14*, *33*
getGenomeAndMask, *3, 4*, *7, 8*, *10*, *12*, 12, *14*, *17*, *33*
getMask, *4*, *8*, *12, 13*, 13, *33*
GRanges, *10*, *12*, *14*, *17*, *40–42*

joinRegions, *5*, *9*, 14, *20*, *24*, *37*

listChrTypes, *10*, 15
localZScore, 16, *27, 28*

maskFromBSGenome, *3, 4*, *8*, *12–14*, 17, *33*
mean, *22*, *26*
meanDistance, 17
meanInRegions, 18
memoise, *12–14*, *17*
mergeRegions, *5*, *9*, *15*, 19, *24*, *37*, *44*

NA, *4*, *7*, *13*, *33*
NULL, *4*, *7*, *13*, *33*
numOverlaps, 20, *22*

overlapGraphicalSummary, 21, *22*
overlapPermTest, *6*, *16*, *21*, 22, *26*
overlapRegions, *5*, *9*, *15*, *20–22*, 23, *37*

par, *21*, *31*
permTest, *6*, *16*, *19*, *21, 22*, 25, *29, 30*, *34*
plot.localZScoreResults, 26
plot.localZScoreResultsList, 27
plot.permTestResults, 28
plot.permTestResultsList, 30
plotRegions, *5*, *9*, *15*, *20*, *24*, 31
print.permTestResults, 32

randomizeRegions, *4*, *8*, *22*, 32, *35, 36*
recomputePermTest, 34
reduce, *14*, *20*
resampleGenome, 35
resampleRegions, *4*, *8*, *33*, 36

sd, *22*, *26*
splitRegions, *5*, *9*, *15*, *20*, *24*, 37
subtractRegions, *5*, *9*, *15*, *20*, *24*, *37*, 38, *44*
summary.permTestResults, 39
summary.permTestResultsList, 39

toDataframe, *4, 5*, *9*, *15*, *20*, *22*, *24*, *33*, *35–37*, 40, *42*
toGRanges, *4, 5*, *9, 10*, *12*, *14–16*, *18–25*, *33*, *35–38*, *40*, 41, *44*

uniqueRegions, 43