Package: bumphunter (via r-universe)

October 1, 2026

Version 1.55.1

Title Bump Hunter

Description Tools for finding bumps in genomic data

Depends R (>= 3.5), S4Vectors (>= 0.9.25), IRanges (>= 2.3.23), Seqinfo, GenomicRanges, foreach, iterators, methods, parallel, locfit

Suggests testthat, RUnit, doParallel, GenomeInfoDb, txdbmaker, org.Hs.eg.db, TxDb.Hsapiens.UCSC.hg19.knownGene

Imports matrixStats, limma, doRNG, BiocGenerics, utils, GenomicFeatures, AnnotationDbi, stats

License Artistic-2.0

LazyData yes

URL https://github.com/rafalab/bumphunter

biocViews DNAMethylation, Epigenetics, Infrastructure, MultipleComparison, ImmunoOncology

RoxygenNote 6.1.1

Config/pak/sysreqs make libbz2-dev liblzma-dev libpng-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

Repository https://bioc.r-universe.dev

Date/Publication 2026-05-25 21:15:24 UTC

RemoteUrl https://github.com/bioc/bumphunter

RemoteRef HEAD

RemoteSha 51b589f2c3be64859df7b50431005b8cec3b72d7

Contents

annotateNearest 2
annotateTranscripts 3
bumphunter 5
clusterMaker 8

dummyData 9
getSegments 10
locfitByCluster 12
loessByCluster 13
matchGenes 14
regionFinder 16
runmedByCluster 17
smoother 19
TT 20

Index **21**

`annotateNearest` *Annotate the results of nearest*

Description

Annotate the results of nearest with more information about the type of match.

Usage

```
annotateNearest(x, subject, annotate = TRUE, ...)
```

Arguments

`x`	The query. An `IRanges` or `GenomicRanges` object, or a `data.frame` with columns for `start`, `end`, and, optionally, `chr` or `seqnames`.
`subject`	The subject. An `IRanges` or `GenomicRanges` object, or a `data.frame` with columns for `start`, `end`, and, optionally, `chr` or `seqnames`.
`annotate`	Whether to annotate the result.
`...`	Arguments passed along to `nearest`.

Details

This function runs `nearest` and then annotates the nearest hit. Note that the nearest subject range to a given query may not be unique and we arbitrarily chose one as done by default by `nearest`.

Value

A data frame with columns `c("distance", "subjectHits", "type", "amountOverlap", "insideDistamce", "size1", "size2")` unless `annotate` is `FALSE`, in which case only the first two columns are returned as an integer matrix.

`dist`	Signed distance to the nearest target. Queries downstream from (i.e. past) their nearest target are given a negative distance.
`subjectHits`	The index of the nearest target.
`type`	one of `c("inside", "cover", "disjoint", "overlap")`.

amountOverlap	The width of the overlap region, if any.
insideDistance	When a query is contained in its nearest target, the signed minimum of the two distances target-start-to-query-start and query-end-to-target-end. The former is taken positive, and the latter, which wins in ties, negative. dist will be 0 in this case.
size1	equals width(x).
size2	equals width(subject).

Author(s)

Harris Jaffee, Peter Murakami and Rafael A. Irizarry

See Also

nearest, matchGenes

Examples

```
query <- GRanges(seqnames = 'chr1', IRanges(c(1, 4, 9), c(5, 7, 10)))
subject <- GRanges('chr1', IRanges(c(2, 2, 10), c(2, 3, 12)))
nearest(query, subject)
distanceToNearest(query, subject)

## showing 'cover' and 'disjoint', and 'amountOverlap'
annotateNearest(query, subject)

## showing 'inside' and 'insideDist', and 'amountOverlap'
annotateNearest(subject, query)
annotateNearest(GRanges('chr1', IRanges(3,3)), GRanges('chr1', IRanges(2,5)))
annotateNearest(GRanges('chr1', IRanges(3,4)), GRanges('chr1', IRanges(2,5)))
annotateNearest(GRanges('chr1', IRanges(4,4)), GRanges('chr1', IRanges(2,5)))
```

---

annotateTranscripts *Annotate transcripts*

Description

Annotate transcripts

Usage

```
annotateTranscripts(txdb, annotationPackage = NULL, by = c("tx","gene"), codingOnly=FALSE, verbose = TR
```

Arguments

txdb	A TxDb database object such as TxDb.Hsapiens.UCSC.hg19.knownGene
annotationPackage	An annotation data package from which to obtain gene/transcript annotation. For example org.Hs.eg.db. If none is provided the function tries to infer it from organism(txdb) and if it can't it proceeds without annotation unless requireAnnotation = TRUE.
by	Should we create a GRanges of transcripts (tx) or genes (gene).
codingOnly	Should we exclude all the non-coding transcripts.
verbose	logical value. If 'TRUE', it writes out some messages indicating progress. If 'FALSE' nothing should be printed.
requireAnnotation	logical value. If 'TRUE' function will stop if no annotation package is successfully loaded.
mappingInfo	a named list with elements 'column', 'keytype' and 'multiVals'. If specified this information will be used with mapIds when mapping the gene ids using annotationPackage. This is useful when working with a txdb object from ENSEMBL or GENCODE among other databases.
simplifyGeneID	logical value. If 'TRUE', gene ids will be shortened to before a dot is present in the id. This is useful for changing GENCODE gene ids to ENSEMBL ids.

Details

This function prepares a GRanges for the matchGenes function. It adds information and in particular adds exons information to each gene/transcript.

Value

A GRanges object with an attribute description set to annotatedTranscripts. The following columns are added. seqinfo is the information returned by seqinfo, CSS is the coding region start, CSE is the coding region end, Tx is the transcript ID used in TxDb, Entrez is the Entrez ID, Gene is the gene symbol, Refseq is the RefSeq annotation, Nexons is the number of exons, Exons is an IRanges with the exon information.

Author(s)

Harris Jaffee and Rafael A. Irizarry. 'mappingInfo' and 'simplifyGeneID' contributed by Leonardo Collado-Torres.

See Also

matchGenes

Examples

```
## Not run:
    library("TxDb.Hsapiens.UCSC.hg19.knownGene")
    genes <- annotateTranscripts(TxDb.Hsapiens.UCSC.hg19.knownGene)

    ##and to avoid guessing the annotation package:
   genes <- annotateTranscripts(TxDb.Hsapiens.UCSC.hg19.knownGene,annotation="org.Hs.eg.db")


## End(Not run)
```

---

bumphunter *Bumphunter*

Description

Estimate regions for which a genomic profile deviates from its baseline value. Originally implemented to detect differentially methylated genomic regions between two populations.

Usage

```
## S4 method for signature 'matrix'
bumphunter(object, design, chr=NULL, pos, cluster=NULL,coef=2, cutoff=NULL, pickCutoff = FALSE, pickCu

bumphunterEngine(mat, design, chr = NULL, pos, cluster = NULL, coef = 2, cutoff = NULL, pickCutoff = FALS
## S3 method for class 'bumps'
print(x, ...)
```

Arguments

object	An object of class matrix.
x	An object of class bumps.
mat	A matrix with rows representing genomic locations and columns representing samples.
design	Design matrix with rows representing samples and columns representing covariates. Regression is applied to each row of mat.
chr	A character vector with the chromosomes of each location.
pos	A numeric vector representing the chromosomal position.
cluster	The clusters of locations that are to be analyzed together. In the case of microarrays, the clusters are many times supplied by the manufacturer. If not available the function clusterMaker can be used to cluster nearby locations.
coef	An integer denoting the column of the design matrix containing the covariate of interest. The hunt for bumps will be only be done for the estimate of this coefficient.

cutoff	A numeric value. Values of the estimate of the genomic profile above the cutoff or below the negative of the cutoff will be used as candidate regions. It is possible to give two separate values (upper and lower bounds). If one value is given, the lower bound is minus the value.
pickCutoff	Should bumphunter attempt to pick a cutoff using the permutation distribution?
pickCutoffQ	The quantile used for picking the cutoff using the permutation distribution.
maxGap	If cluster is not provided this maximum location gap will be used to define cluster via the clusterMaker function.
nullMethod	Method used to generate null candidate regions, must be one of 'bootstrap' or 'permutation' (defaults to 'permutation'). However, if covariates in addition to the outcome of interest are included in the design matrix (ncol(design)>2), the 'permutation' approach is not recommended. See vignette and original paper for more information.
smooth	A logical value. If TRUE the estimated profile will be smoothed with the smoother defined by smoothFunction
smoothFunction	A function to be used for smoothing the estimate of the genomic profile. Two functions are provided by the package: loessByCluster and runmedByCluster.
useWeights	A logical value. If TRUE then the standard errors of the point-wise estimates of the profile function will be used as weights in the loess smoother loessByCluster. If the runmedByCluster smoother is used this argument is ignored.
B	An integer denoting the number of resamples to use when computing null distributions. This defaults to 0. If permutations is supplied that defines the number of permutations/bootstraps and B is ignored.
permutations	is a matrix with columns providing indexes to be used to scramble the data and create a null distribution when nullMethod is set to permutations. If the bootstrap approach is used this argument is ignored. If this matrix is not supplied and B>0 then these indexes are created using the function sample.
verbose	logical value. If TRUE, it writes out some messages indicating progress. If FALSE nothing should be printed.
...	further arguments to be passed to the smoother functions.

Details

This function performs the bumphunting approach described by Jaffe et al. International Journal of Epidemiology (2012). The main output is a table of candidate regions with permutation or bootstrap-based family-wide error rates (FWER) and p-values assigned.

The general idea is that for each genomic location we have a value for several individuals. We also have covariates for each individual and perform regression. This gives us one estimate of the coefficient of interest (a common example is case versus control). These estimates are then (optionally) smoothed. The smoothing occurs in clusters of locations that are 'close enough'. This gives us an estimate of a genomic profile that is 0 when uninteresting. We then take values above (in absolute value) cutoff as candidate regions. Permutations can then performed to create null distributions for the candidate regions.

The simplest way to use permutations or bootstraps to create a null distribution is to set B. If the number of samples is large this can be set to a large number, such as 1000. Note that this will be slow

and we have therefore provided parallelization capabilities. In cases were the user wants to define the permutations or bootstraps, for example cases in which all possible permutations/boostraps can be enumerated, these can be supplied via the `permutations` argument.

Uncertainty is assessed via permutations or bootstraps. Each of the `B` permutations/bootstraps will produce an estimated 'null profile' from which we can define 'null candidate regions'. For each observed candidate region we determine how many null regions are 'more extreme' (longer and higher average value). The 'p.value' is the percent of candidate regions obtained from the permutations/boostraps that are as extreme as the observed region. These p-values should be interpreted with care as the theoretical properties are not well understood. The 'fwer' is the proportion of permutations/bootstraps that had at least one region as extreme as the observed region. We compute p.values and FWER for the area of the regions (as opposed to length and value as a pair) as well. Note that for cases with more than one covariate the permutation approach is not generally recommended; the `nullMethod` argument will coerce to 'bootstrap' in this scenario. See vignette and original paper for more information.

Parallelization is implemented through the foreach package.

Value

An object of class `bumps` with the following components:

`tab`	The table with candidate regions and annotation for these.
`coef`	The single loci coefficients.
`fitted`	The estimated genomic profile used to determine the regions.
`pvaluesMarginal`	marginal p-value for each genomic location.
`null`	The null distribution.
`algorithm`	details on the algorithm.

Author(s)

Rafael A. Irizarry, Martin J. Aryee, Kasper D. Hansen, and Shan Andrews.

References

Jaffe AE, Murakami P, Lee H, Leek JT, Fallin MD, Feinberg AP, Irizarry RA (2012) Bump hunting to identify differentially methylated regions in epigenetic epidemiology studies. International Journal of Epidemiology 41(1):200-9.

Examples

```
dat <- dummyData()
# Enable parallelization
require(doParallel)
registerDoParallel(cores = 2)
# Find bumps
bumps <- bumphunter(dat$mat, design=dat$design, chr=dat$chr, pos=dat$pos,
                    cluster=dat$cluster, coef=2, cutoff= 0.28, nullMethod="bootstrap",
                    smooth=TRUE, B=250, verbose=TRUE,
```

```
                    smoothFunction=loessByCluster)
bumps
# cleanup, for Windows
bumphunter:::foreachCleanup()
```

`clusterMaker` *Make clusters of genomic locations based on distance*

Description

Genomic locations are grouped into clusters based on distance: locations that are close to each other are assigned to the same cluster. The operation is performed on each chromosome independently.

Usage

```
clusterMaker(chr, pos, assumeSorted = FALSE, maxGap = 300)
boundedClusterMaker(chr, pos, assumeSorted = FALSE,
                    maxClusterWidth = 1500, maxGap = 500)
```

Arguments

`chr`	A vector representing chromosomes. This is usually a character vector, but may be a factor or an integer.
`pos`	A numeric vector with genomic locations.
`assumeSorted`	This is a statement that the function may assume that the vector `pos` is sorted (within each `chr`). Allowing the function to make this assumption may increase the speed of the function slightly.
`maxGap`	An integer. Genomic locations within maxGap from each other are placed into the same cluster.
`maxClusterWidth`	An integer. A cluster large than this width is broken into subclusters.

Details

The main purpose of the function is to genomic location into clusters that are close enough to perform operations such as smoothing. A genomic location is a combination of a chromosome (`chr`) and an integer position (`pos`). Specifically, genomic intervals are not handled by this function.

Each chromosome is clustered independently from each other. Within each chromosome, clusters are formed in such a way that two positions belong to the same cluster if they are within `maxGap` of each other.

Value

A vector of integers to be interpreted as IDs for the clusters, such that two genomic positions with the same cluster ID is in the same cluster. Each genomic position receives one integer ID.

Author(s)

Rafael A. Irizarry, Hector Corrada Bravo

Examples

```
N <- 1000
chr <- sample(1:5, N, replace=TRUE)
pos <- round(runif(N, 1, 10^5))
o <- order(chr, pos)
chr <- chr[o]
pos <- pos[o]
regionID <- clusterMaker(chr, pos)
regionID2 <- boundedClusterMaker(chr, pos)
```

---

dummyData *Generate dummy data for use with bumphunter functions*

Description

This function generates a small dummy dataset representing samples from two different groups (cases and controls) that is used in bumphunter examples.

Usage

```
dummyData(n1 = 5, n2 = 5, sd = 0.2, l = 100, spacing = 100,
          clusterSpacing=1e5, numClusters=5)
```

Arguments

n1	Number of samples in group 1 (controls)
n2	Number of samples in group 2 (cases)
sd	Within group standard deviation to be used when simulating data
l	The number of genomic locations for which to simulate data
spacing	The average spacing between locations. The actual locations have a random component so the actual spacing will be non-uniform
clusterSpacing	The spacing between clusters. (Specifically, the spacing between the first location in each cluster.)
numClusters	Divide the genomic locations into this number of clusters, each of which will contain locations spaced spacing bp apart.

Value

A list containing data that can be used with various bumphunter functions.

mat	A simulated data matrix with rows representing genomic locations and columns representing samples.
design	Design matrix with rows representing samples and columns representing covariates.
chr	A character vector with the chromosomes of each location.
pos	A numeric vector representing the chromosomal position.
cluster	A vector representing the cluster of each locations
n1	Number of samples in cluster 1
n2	Number of samples in cluster 2

Author(s)

Martin J. Aryee

Examples

```
dat <- dummyData()
names(dat)
head(dat$pos)
```

---

getSegments *Segment a vector into positive, zero, and negative regions*

Description

Given two cutoffs, L and U, this function divides a numerical vector into contiguous parts that are above U, between L and U, and below L.

Usage

```
getSegments(x, f = NULL, cutoff = quantile(abs(x), 0.99),
            assumeSorted = FALSE, verbose = FALSE)
```

Arguments

x	A numeric vector.
f	A factor used to pre-divide x into pieces. Each piece is then segmented based on the cutoff. Setting this to NULL says that there is no pre-division. Often, clusterMaker is used to define this factor.
cutoff	a numeric vector of length either 1 or 2. If length is 1, U (see details) will be cutoff and L will be -cutoff. Otherwise it specifies L and U. The function will furthermore always use the minimum of cutoff for L and the maximum for U.

assumeSorted	This is a statement that the function may assume that the vector f is sorted. Allowing the function to make this assumption may increase the speed of the function slightly.
verbose	Should the function be verbose?

Details

This function is used to find the indexes of the 'bumps' in functions such as bumphunter.

x is a numeric vector, which is converted into three levels depending on whether x>=U ('up'), L<x<U ('zero') or x<=L ('down'), with L and U coming from cutoff. We assume that adjacent entries in x are next to each other in some sense. Segments, consisting of consecutive indices into x (ie. values between 1 and length(x)), are formed such that all indices in the same segment belong to the same level of f and have the same discretized value of x.

In other words, we can use getSegments to find runs of x belonging to the same level of f and with all of the values of x either above U, between L and U, or below L.

Value

A list with three components, each a list of indices. Each component of these lists represents a segment and this segment is represented by a vector of indices into the original vectors x and f.

upIndex:	a list with each entry an index of contiguous values in the same segment. These segments have values of x above U.
dnIndex:	a list with each entry an index of contiguous values in the same segment. These segments have values of x below L.
zeroIndex:	a list with each entry an index of contiguous values in the same segment. These segments have values of x between L and U.

Author(s)

Rafael A Irizarry and Kasper Daniel Hansen

See Also

clusterMaker

Examples

```
x <- 1:100
y <- sin(8*pi*x/100)
chr <- rep(1, length(x))
indexes <- getSegments(y, chr, cutoff=0.8)
plot(x, y, type="n")
for(i in 1:3){
   ind <- indexes[[i]]
   for(j in seq(along=ind)) {
      k <- ind[[j]]
      text(x[k], y[k], j, col=i)
   }
```

```
}
abline(h=c(-0.8,0.8))
```

`locfitByCluster` *Apply local regression smoothing to values within each spatially-defined cluster.*

Description

Local regression smoothing with a gaussian kernal, is applied independently to each cluster of genomic locations. Locations within the same cluster are close together to warrant smoothing across neighbouring locations.

Usage

```
locfitByCluster(y, x = NULL, cluster, weights = NULL, minNum = 7,
                bpSpan = 1000, minInSpan = 0, verbose = TRUE)
```

Arguments

`y`	A vector or matrix of values to be smoothed. If a matrix, each column represents a sample.
`x`	The genomic location of the values in y
`cluster`	A vector indicating clusters of locations. A cluster is typically defined as a region that is small enough that it makes sense to smooth across neighbouring locations. Smoothing will only be applied within a cluster, not across locations from different clusters.
`weights`	weights used by the locfit smoother
`minNum`	Clusters with fewer than `minNum` locations will not be smoothed
`bpSpan`	The span used when locfit smoothing. (Expressed in base pairs.)
`minInSpan`	Only smooth the region if there are at least this many locations in the span.
`verbose`	Boolean. Should progress be reported?

Details

This function is typically called by `smoother`, which is in turn called by `bumphunter`.

Value

`fitted`	The smoothed data values
`smoothed`	A boolean vector indicating whether a given position was smoothed
`smoother`	always set to 'locfit'.

Author(s)

Rafael A. Irizarry and Kasper D. Hansen

See Also

smoother, runmedByCluster, loessByCluster

Examples

```
dat <- dummyData()
smoothed <- locfitByCluster(y=dat$mat[,1], cluster=dat$cluster, bpSpan = 1000,
                            minNum=7, minInSpan=5)
```

---

loessByCluster *Apply loess smoothing to values within each spatially-defined cluster.*

Description

Loess smoothing is applied independently to each cluster of genomic locations. Locations within the same cluster are close together to warrant smoothing across neighbouring locations.

Usage

```
loessByCluster(y, x = NULL, cluster, weights = NULL, bpSpan = 1000,
               minNum = 7, minInSpan = 5, maxSpan = 1, verbose = TRUE)
```

Arguments

y	A vector or matrix of values to be smoothed. If a matrix, each column represents a sample.
x	The genomic location of the values in y
cluster	A vector indicating clusters of locations. A cluster is typically defined as a region that is small enough that it makes sense to smooth across neighbouring locations. Smoothing will only be applied within a cluster, not across locations from different clusters.
weights	weights used by the loess smoother
bpSpan	The span used when loess smoothing. (Expressed in base pairs.)
minNum	Clusters with fewer than minNum locations will not be smoothed
minInSpan	Only smooth the region if there are at least this many locations in the span.
maxSpan	The maximum span. Spans greater than this value will be capped.
verbose	Boolean. Should progress be reported?

Details

This function is typically called by smoother, which is in turn called by bumphunter.

Value

fitted	The smoothed data values
smoothed	A boolean vector indicating whether a given position was smoothed
smoother	always set to 'loess'.

Author(s)

Rafael A. Irizarry

See Also

smoother, runmedByCluster, locfitByCluster

Examples

```
dat <- dummyData()
smoothed <- loessByCluster(y=dat$mat[,1], cluster=dat$cluster, bpSpan = 1000,
                           minNum=7, minInSpan=5, maxSpan=1)
```

---

matchGenes *Find and annotate closest genes to genomic regions*

Description

Find and annotate closest genes to genomic regions

Usage

```
matchGenes(x, subject, type = c("any", "fiveprime"), promoterDist = 2500, skipExons = FALSE, verbose = T
```

Arguments

x	An IRanges or GenomicRanges object, or a data.frame with columns for start, end, and, optionally, chr or seqnames.
subject	An GenomicRanges object containing transcripts or genes that have been annotated by the function annotateTranscripts.
promoterDist	Anything within this distance to the transcription start site (TSE) will be considered a promoter.
type	Should the distance be computed to any part of the transcript or the five prime end.
skipExons	Should the annotation of exons be skipped. Skipping this part makes the code slightly faster.
verbose	logical value. If 'TRUE', it writes out some messages indicating progress. If 'FALSE' nothing should be printed.

Details

This function runs nearest and then annotates the the relationship between the region and the transcript/gene that is closest. Many details are provided on this relationship as described in the next section.

Value

A data frame with one row for each query and with columns c("name", "annotation", "description", "region", "distance", "subregion", "insideDistance", "exonnumber", "nexons", "UTR", "strand", "geneL", "codingL","Entrez","subhectHits"). The first column is the _gene_ nearest the query, by virtue of it owning the transcript determined (or chosen by nearest) to be nearest the query. Note that the nearest gene to a given query, in column 3, may not be unique and we arbitrarily chose one as done by default by nearest.

The "distance" column is the distance from the query to the 5' end of the nearest transcript, so may be different from the distance computed by nearest to that transcript, as a range.

name	Symbol of nearest gene
annotation	RefSeq ID
description	a factor with levels c("upstream", "promoter","overlaps 5'", "inside intron", "inside exon", "covers exon(s)","overlaps exon upstream", "overlaps exon downstream","overlaps two exons", "overlaps 3'", "close to 3'", "downstream","covers")
region	a factor with levels c("upstream", "promoter", "overlaps 5'","inside", "overlaps 3'", "close to 3'", "downstream", "covers")
distance	distance before 5' end of gene
subregion	a factor with levels c("inside intron", "inside exon","covers exon(s)", "overlaps exon upstream", "overlaps exon downstream","overlaps two exons")
insideDistance	distance past 5' end of gene
exonnumber	which exon
nexons	number of exons
UTR	a factor with levels c("inside transcription region", "5' UTR","overlaps 5' UTR", "3'UTR", "overlaps 3'UTR", "covers transcription region")
strand	"+" or "-"
geneL	the gene length
codingL	the coding length
Entrez	Entrez ID of closest gene
subjectHits	Index in subject of hit

Author(s)

Harris Jaffee, Peter Murakami and Rafael A. Irizarry

See Also

annotateNearest, annotateTranscripts

Examples

```
## Not run:
  islands=makeGRangesFromDataFrame(read.delim("http://rafalab.jhsph.edu/CGI/model-based-cpg-islands-hg19.txt")
   library("TxDb.Hsapiens.UCSC.hg19.knownGene")
   genes <- annotateTranscripts(TxDb.Hsapiens.UCSC.hg19.knownGene)
   tab<- matchGenes(islands,genes)

## End(Not run)
```

`regionFinder` *Find non-zero regions in vector*

Description

Find regions for which a numeric vector is above (or below) predefined thresholds.

Usage

```
regionFinder(x, chr, pos, cluster = NULL, y = x, summary = mean,
             ind = seq(along = x), order = TRUE, oneTable = TRUE,
             maxGap = 300, cutoff=quantile(abs(x), 0.99),
             assumeSorted = FALSE, verbose = TRUE)
```

Arguments

`x`	A numeric vector.
`chr`	A character vector with the chromosomes of each location.
`pos`	A numeric vector representing the genomic location.
`cluster`	The clusters of locations that are to be analyzed together. In the case of microarrays, the cluster is many times supplied by the manufacturer. If not available the function `clusterMaker` can be used.
`y`	A numeric vector with same length as `x` containing values to be averaged for the region summary. See details for more.
`summary`	The function to be used to construct a summary of the `y` values for each region.
`ind`	an optional vector specifying a subset of observations to be used when finding regions.
`order`	if `TRUE` then the resulting tables are ordered based on area of each region. Area is defined as the absolute value of the summarized `y` times the number of features in the regions.
`oneTable`	if `TRUE` only one results table is returned. Otherwise, two tables are returned: one for the regions with positive values and one for the negative values.
`maxGap`	If cluster is not provided this number will be used to define clusters via the `clusterMaker` function.

cutoff	This argument is passed to getSegments. It represents the upper (and optionally the lower) cutoff for x.
assumeSorted	This argument is passed to getSegments and clusterMaker.
verbose	Should the function be verbose?

Details

This function is used in the final steps of bumphunter. While bumphunter does many things, such as regression and permutation, regionFinder simply finds regions that are above a certain threshold (using getSegments) and summarizes them. The regions are found based on x and the summarized values are based on y (which by default equals x). The summary is used for the ranking so one might, for example, use t-tests to find regions but summarize using effect sizes.

Value

If oneTable is FALSE it returns two tables otherwise it returns one table. The rows of the table are regions. Information on the regions is included in the columns.

Author(s)

Rafael A Irizarry

See Also

bumphunter for the main usage of this function, clusterMaker for the typical input to the cluster argument and getSegments for a function used within regionFinder.

Examples

```
x <- seq(1:1000)
y <- sin(8*pi*x/1000) + rnorm(1000, 0, 0.2)
chr <- rep(c(1,2), each=length(x)/2)
tab <- regionFinder(y, chr, x, cutoff=0.8)
print(tab[tab$L>10,])
```

---

runmedByCluster *Apply running median smoothing to values within each spatially-defined cluster*

Description

Running median smoothing is applied independently to each cluster of genomic locations. Locations within the same cluster are close together to warrant smoothing across neighbouring locations.

Usage

```
runmedByCluster(y, x = NULL, cluster, weights = NULL, k = 5,
                endrule = "constant", verbose = TRUE)
```

Arguments

y	A vector or matrix of values to be smoothed. If a matrix, each column represents a sample.
x	The genomic location of the values in y.
cluster	A vector indicating clusters of locations. A cluster is typically defined as a region that is small enough that it makes sense to smooth across neighbouring locations. Smoothing will only be applied within a cluster, not across locations from different clusters.
weights	weights used by the smoother.
k	integer width of median window; must be odd. See runmed
endrule	character string indicating how the values at the beginning and the end (of the data) should be treated. See runmed.
verbose	Boolean. Should progress be reported?

Details

This function is typically called by smoother, which is in turn called by bumphunter.

Value

fitted	The smoothed data values
smoothed	A boolean vector indicating whether a given position was smoothed
spans	The span used by the loess smoother. One per cluster.
clusterL	The number of locations in each cluster.
smoother	always set to 'runmed'.

Author(s)

Rafael A. Irizarry

See Also

smoother, loessByCluster. Also see runmed.

Examples

```
dat <- dummyData()
smoothed <- runmedByCluster(y=dat$mat[,1], cluster=dat$cluster,
                            k=5, endrule="constant")
```

smoother *Smooth genomic profiles*

Description

Apply smoothing to values typically representing the difference between two populations across genomic regions.

Usage

```
smoother(y, x = NULL, cluster, weights = NULL, smoothFunction,
         verbose = TRUE, ...)
```

Arguments

y	A vector or matrix of values to be smoothed. If a matrix, each column represents a sample.
x	The genomic location of the values in y
cluster	A vector indicating clusters of locations. A cluster is typically defined as a region that is small enough that it makes sense to smooth across neighbouring locations. Smoothing will only be applied within a cluster, not across locations from different clusters
weights	weights used by the smoother.
smoothFunction	A function to be used for smoothing the estimate of the genomic profile. Two functions are provided by the package: loessByCluster and runmedByCluster.
verbose	Boolean. Should progress be reported?
...	Further arguments to be passed to smoothFunction

Details

This function is typically called by bumphunter prior to identifying candidate bump regions. Smoothing is carried out within regions defined by the cluster argument.

Value

fitted	The smoothed data values
smoothed	A boolean vector indicating whether a given position was smoothed
spans	The span used by the loess smoother. One per cluster.
clusterL	The number of locations in each cluster.
smoother	The name of the smoother used

Author(s)

Rafael A. Irizarry and Martin J. Aryee

See Also

loessByCluster, runmedByCluster

Examples

```
dat <- dummyData()

# Enable parallelization
require(doParallel)
registerDoParallel(cores = 2)

## loessByCluster
smoothed <- smoother(y=dat$mat[,1], cluster=dat$cluster, smoothFunction=loessByCluster,
                       bpSpan = 1000, minNum=7, minInSpan=5, maxSpan=1)
## runmedByCluster
smoothed <- smoother(y=dat$mat[,1], cluster=dat$cluster, smoothFunction=runmedByCluster,
                     k=5, endrule="constant")
# cleanup, for Windows
bumphunter:::foreachCleanup()
```

TT *Example data*

Description

Example data

Format

A list with three components.

txdb Has a TxDb example.

org has an Org DB example.

transcripts has example transcripts output.

Index

∗ **datasets**
TT, 20

annotateNearest, 2
annotateTranscripts, 3, *14*

boundedClusterMaker *(*clusterMaker*)*, 8
bumphunter, 5, *11–13*, *17*, *18*
bumphunter,matrix-method *(*bumphunter*)*, 5
bumphunter-methods *(*bumphunter*)*, 5
bumphunterEngine *(*bumphunter*)*, 5

clusterMaker, *5*, *6*, 8, *10*, *11*, *16*, *17*

dummyData, 9

getSegments, 10, *17*

locfitByCluster, 12, *14*
loessByCluster, 13, *13*, *18*, *20*

mapIds, *4*
matchGenes, *3*, *4*, 14

nearest, *2*, *3*, *15*

print.bumps *(*bumphunter*)*, 5

regionFinder, 16
runmed, *18*
runmedByCluster, *13*, *14*, 17, *20*

seqinfo, *4*
smoother, *12–14*, *18*, 19

TT, 20